



# MICROSERVICES IN THREE WEEKS

**WEEK 3: DATA**

**Sam Newman**

**This is week three of three!**

**Each session is three hours long**

**Each session is three hours long**

**We will have two 10-15min breaks**

**Please ask questions using the Q&A widget**

## WHAT WE'LL COVER

# Brief Recap

## WHAT WE'LL COVER

### Brief Recap

### App Decomposition (continued)

## WHAT WE'LL COVER

Brief Recap

App Decomposition (continued)

Hiding Data

## WHAT WE'LL COVER

Brief Recap

App Decomposition (continued)

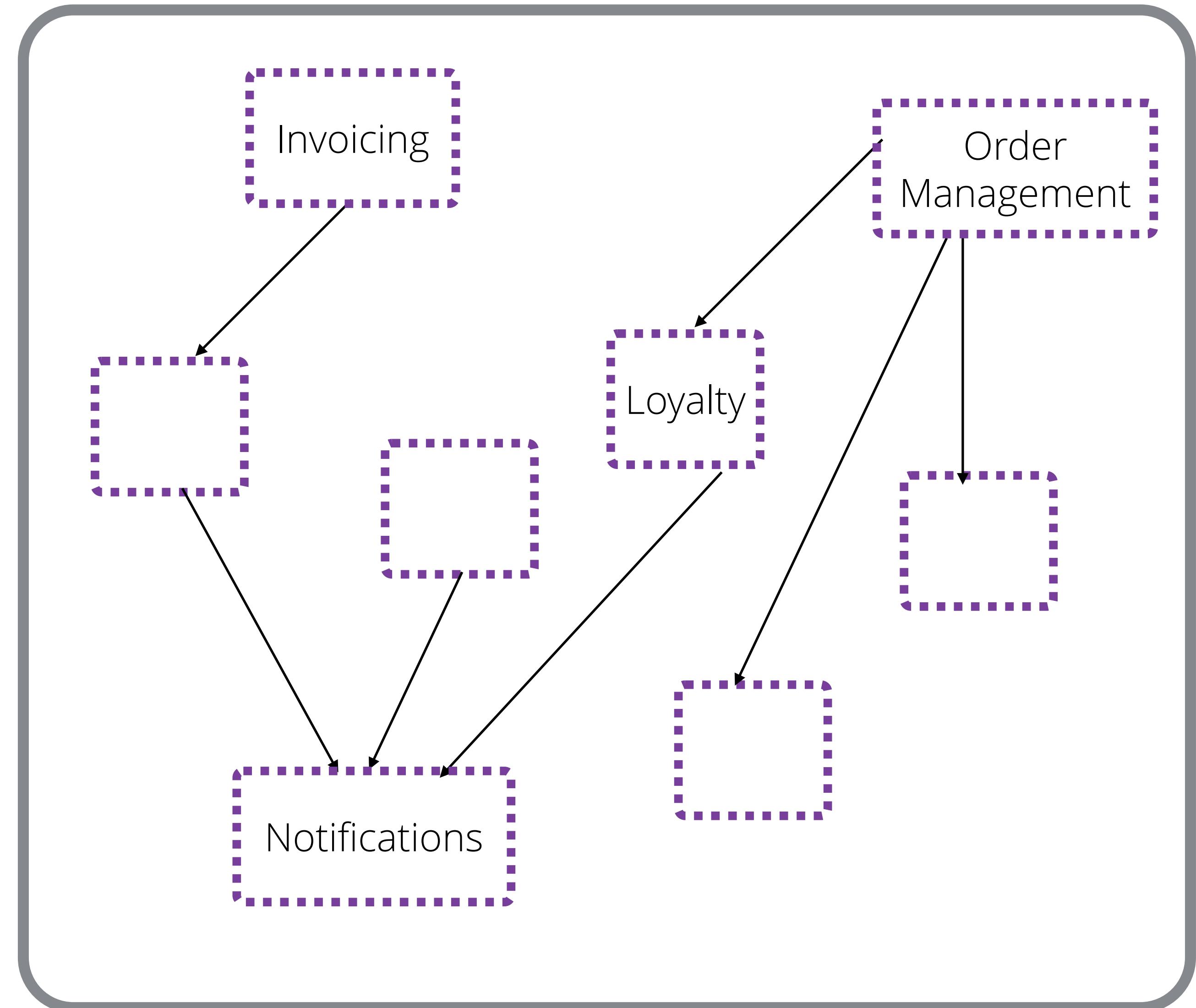
Hiding Data

Database Decomposition

# 1/4 Recap

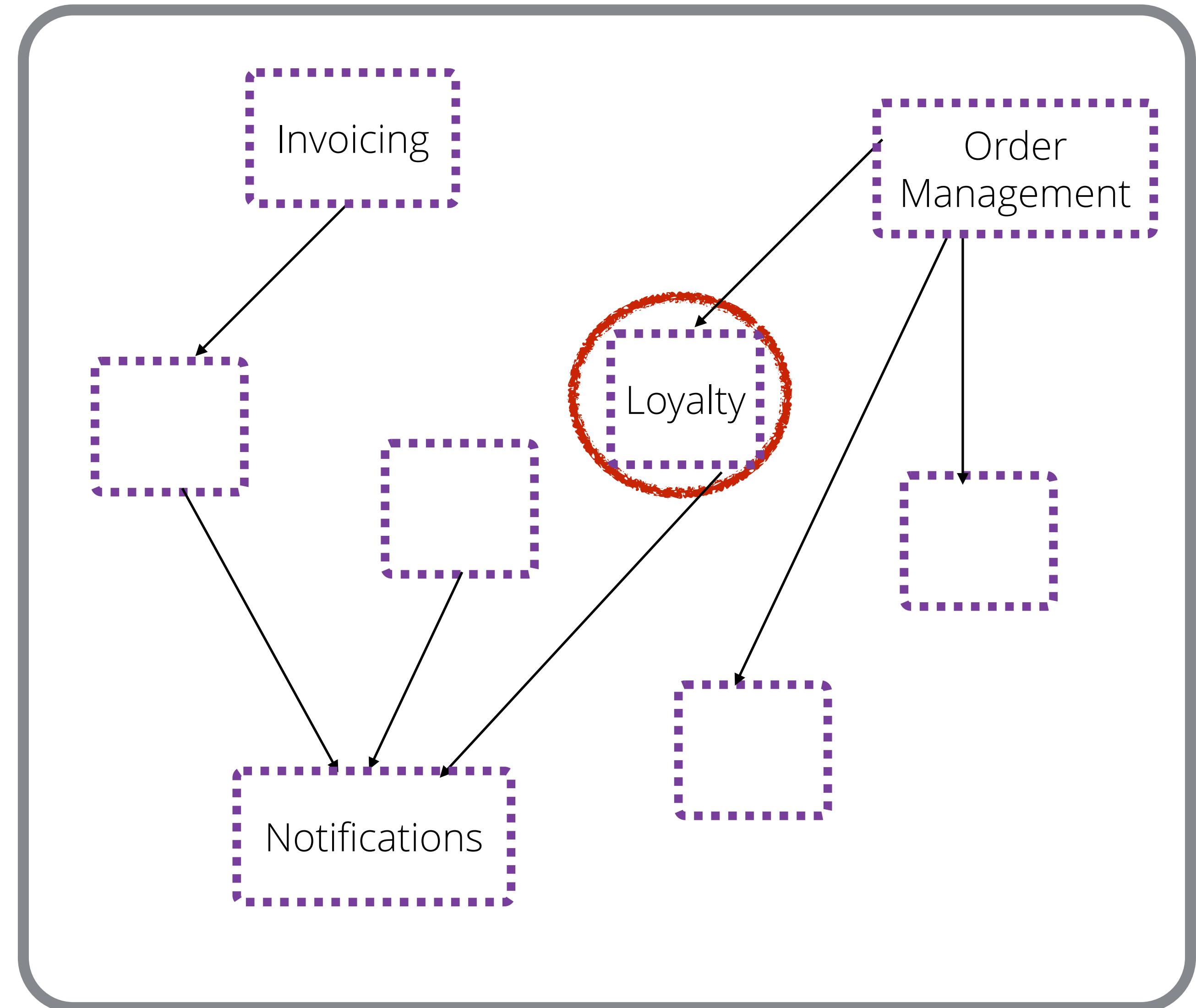
## WHAT ABOUT OTHER FUNCTIONALITY?

Can use “branch by abstraction” to help extraction of functionality deeper in the monolith



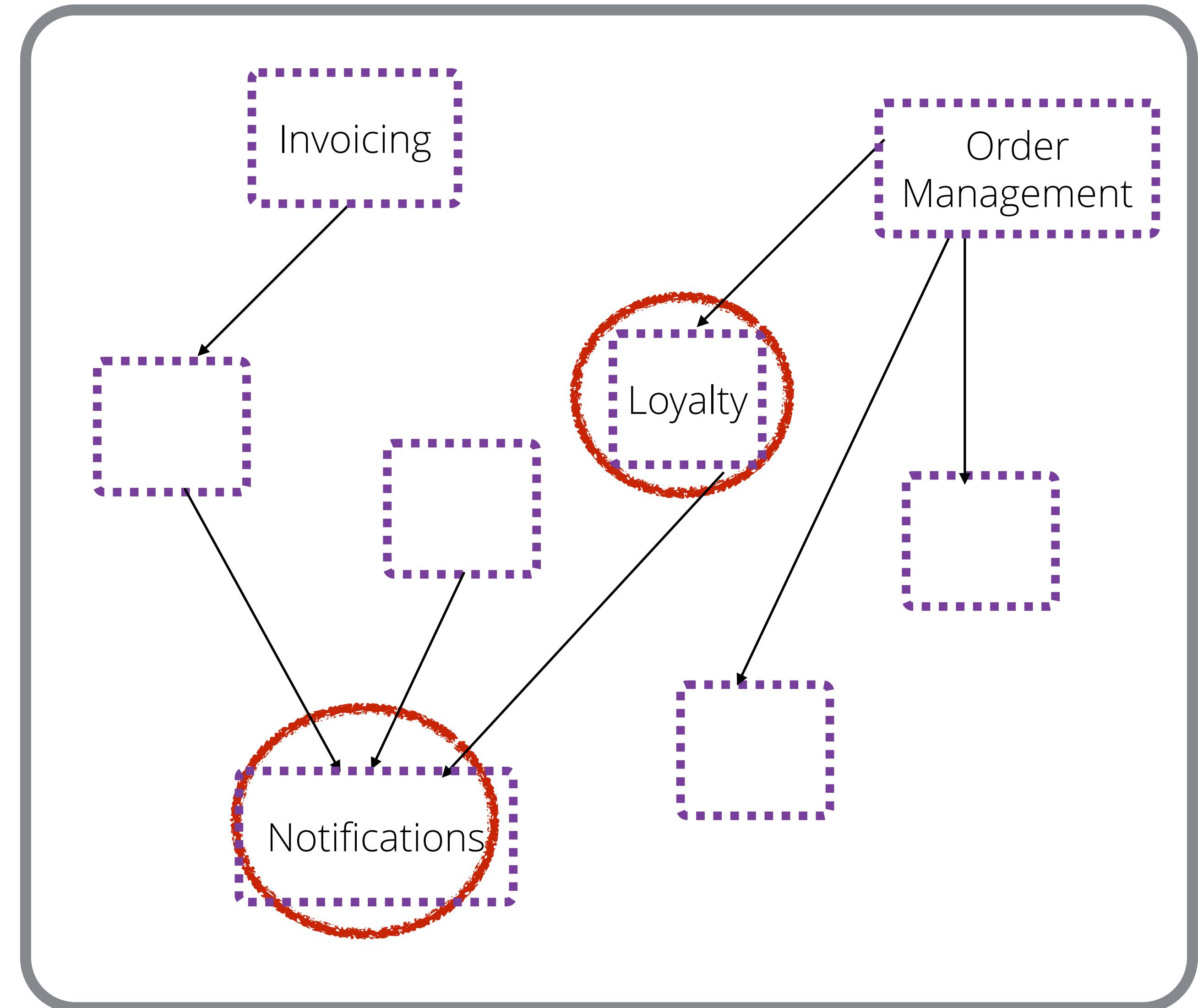
## WHAT ABOUT OTHER FUNCTIONALITY?

Can use “branch by abstraction” to help extraction of functionality deeper in the monolith

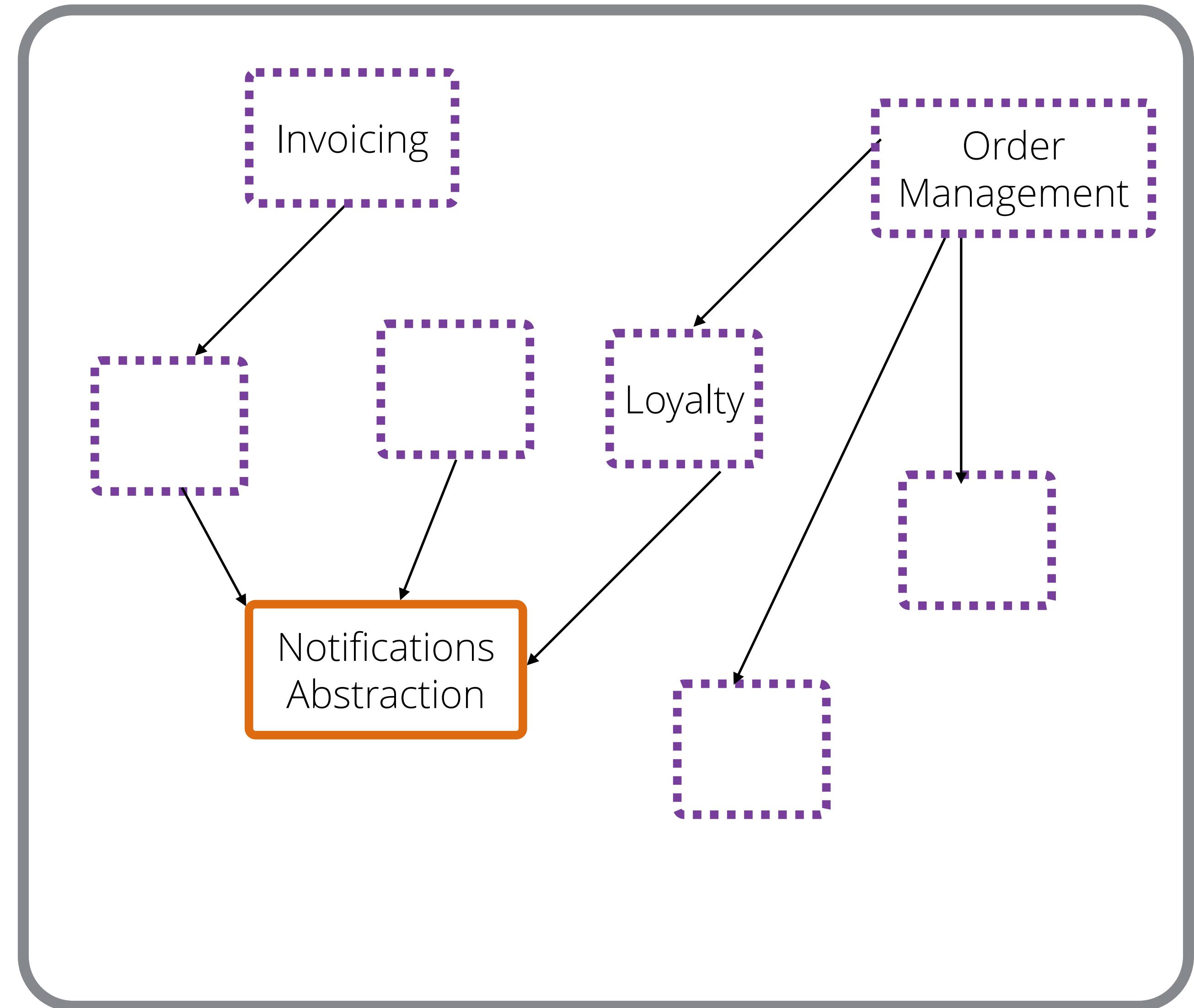


## WHAT ABOUT OTHER FUNCTIONALITY?

Can use “branch by abstraction” to help extraction of functionality deeper in the monolith

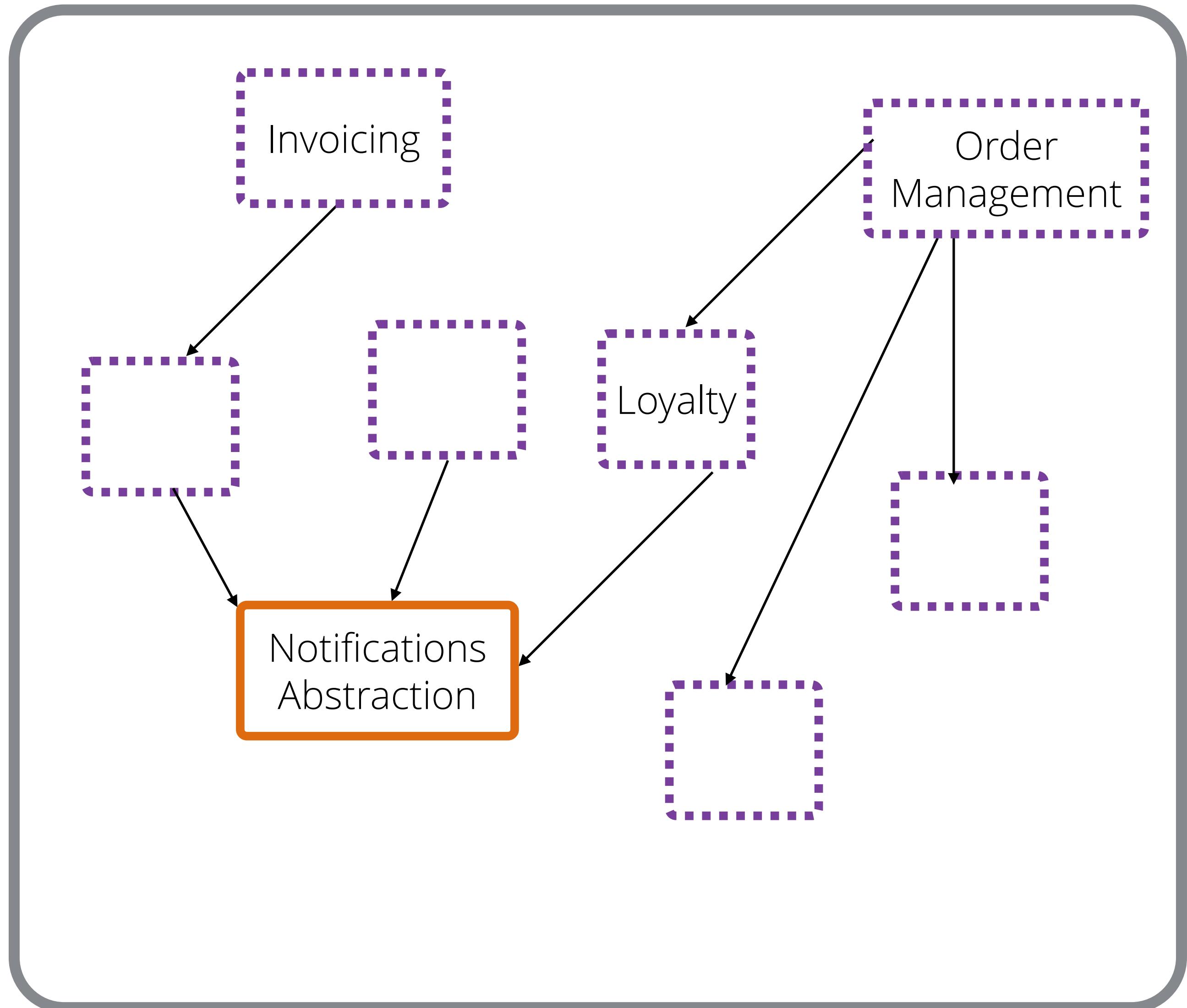


# BRANCH BY ABSTRACTION



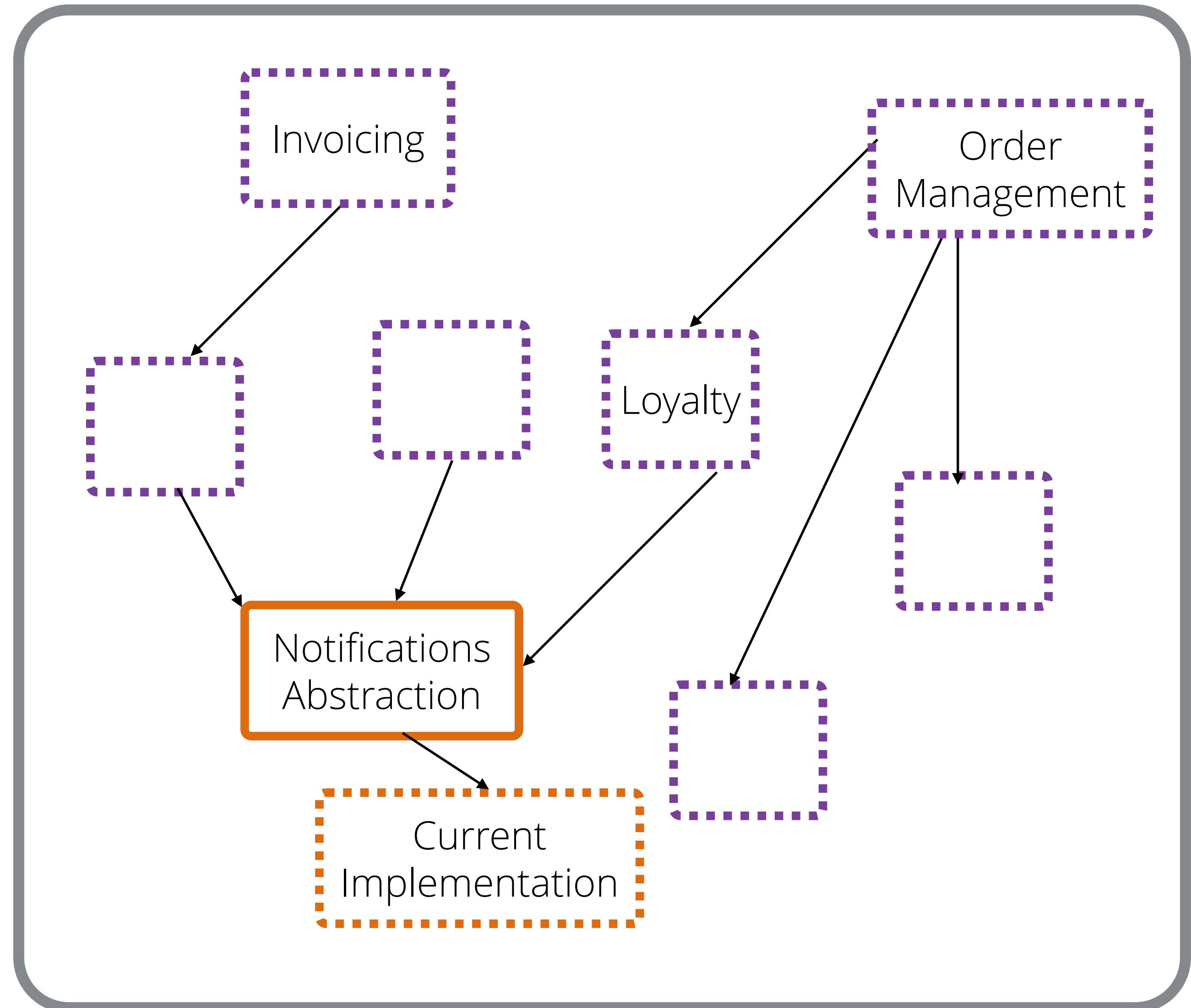
## BRANCH BY ABSTRACTION

Gives you an abstraction point which can be used to “hide” ongoing development



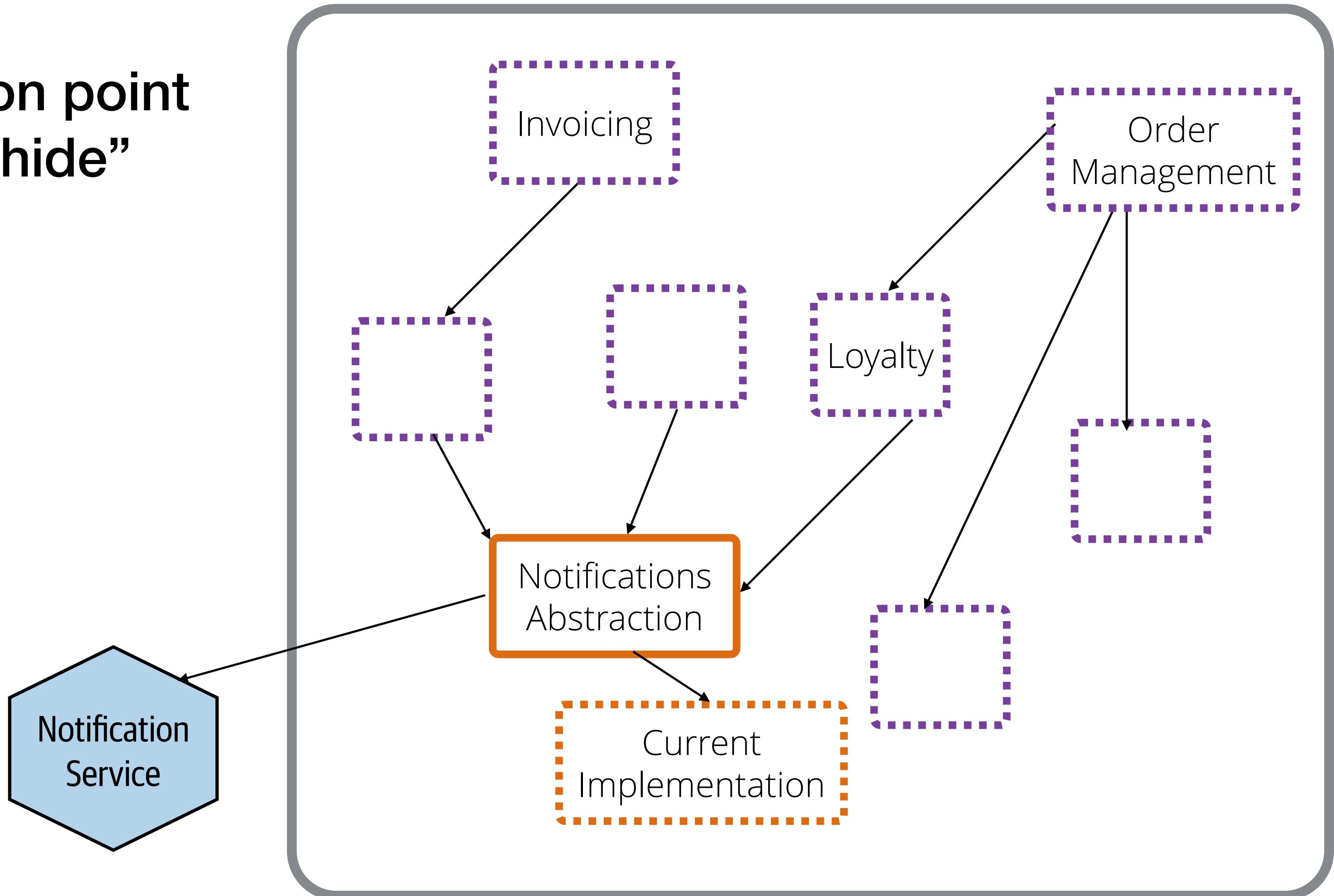
## BRANCH BY ABSTRACTION

Gives you an abstraction point which can be used to “hide” ongoing development



## BRANCH BY ABSTRACTION

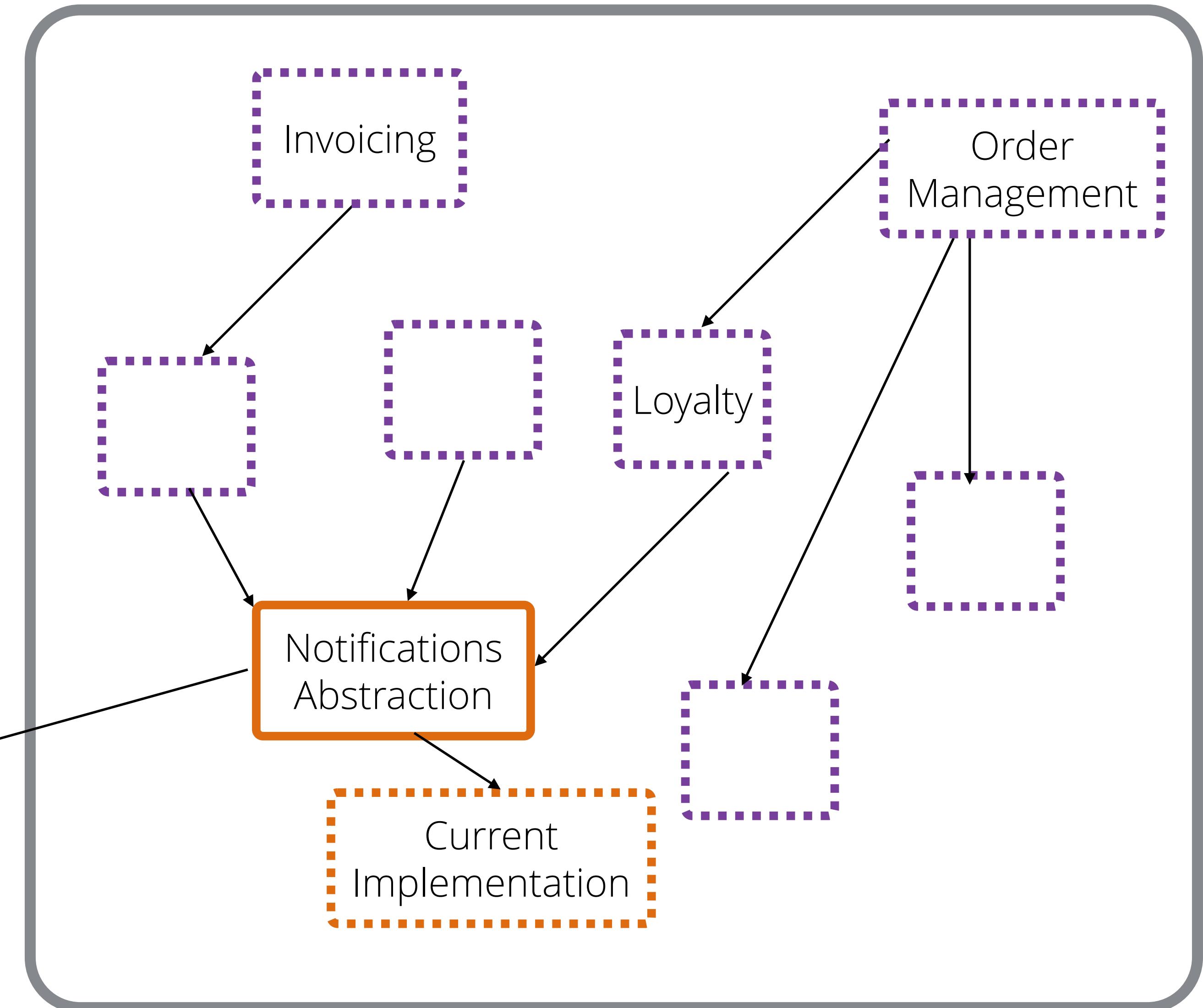
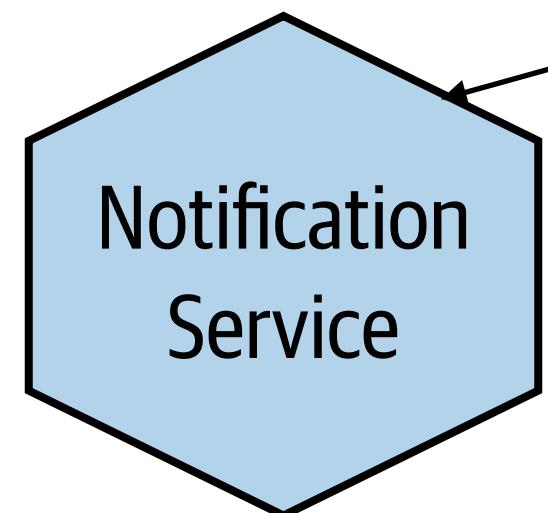
Gives you an abstraction point which can be used to “hide” ongoing development



## BRANCH BY ABSTRACTION

Gives you an abstraction point which can be used to “hide” ongoing development

Can also be used as a way to toggle between implementations



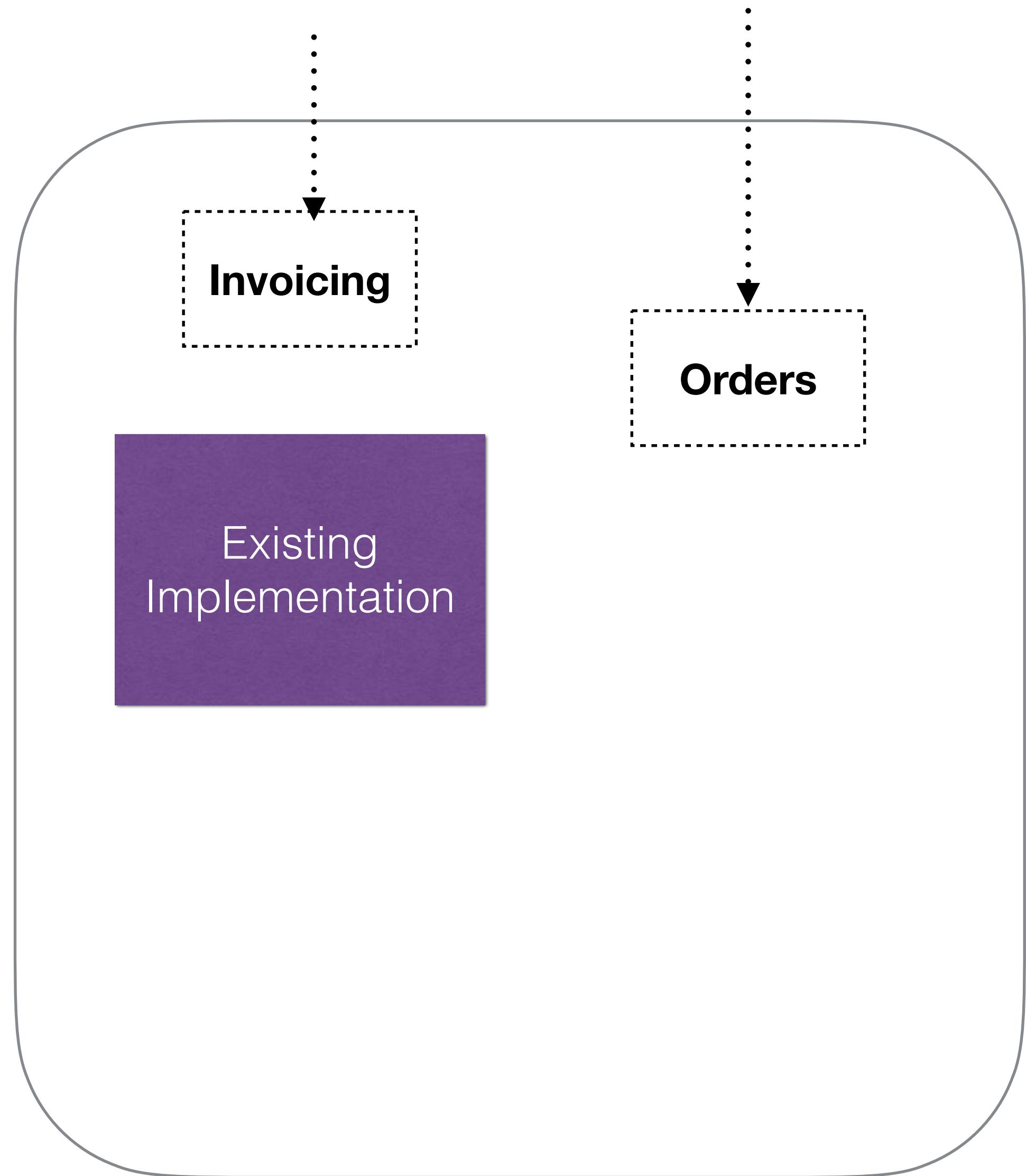
## EXAMPLE



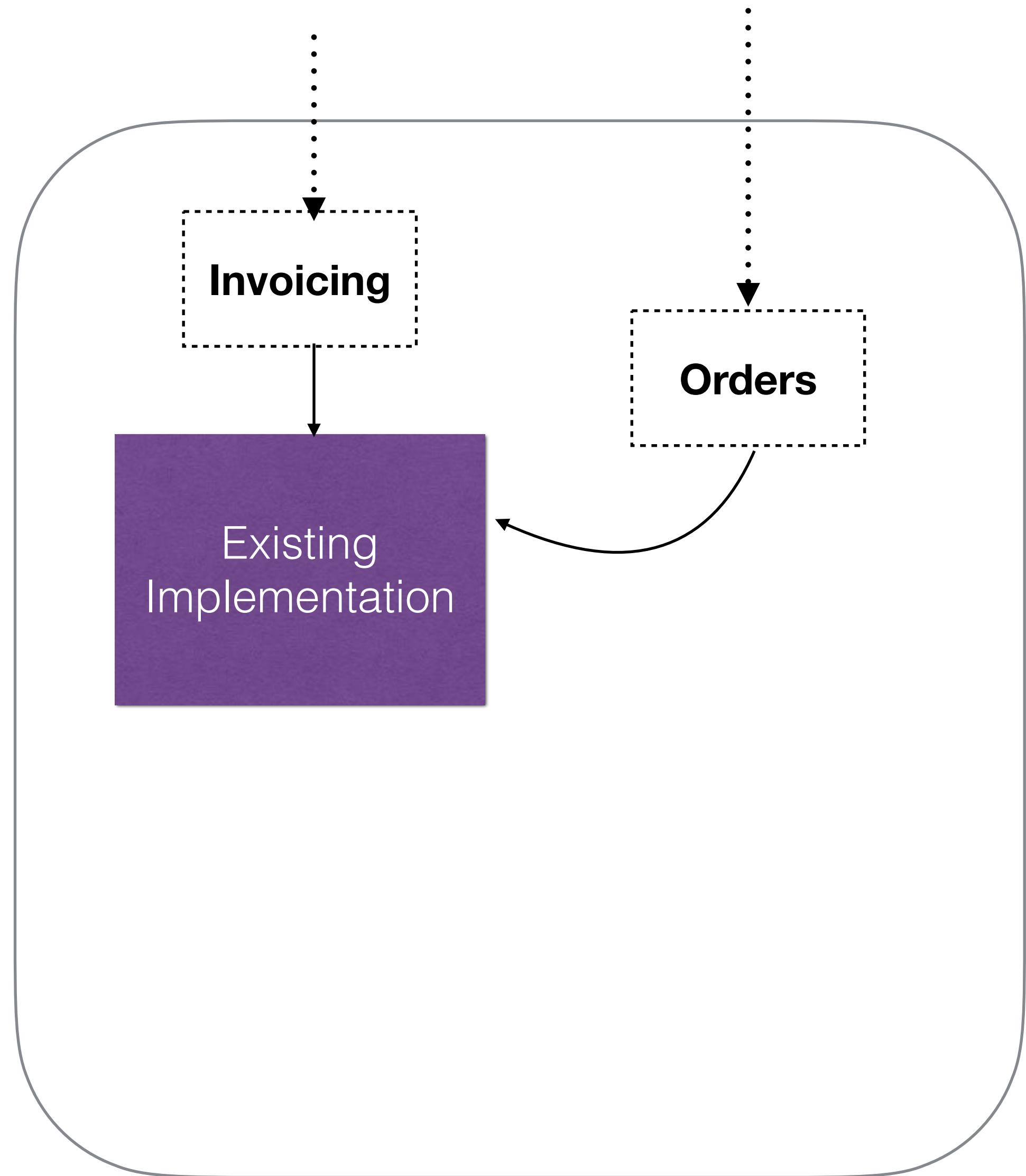
## EXAMPLE



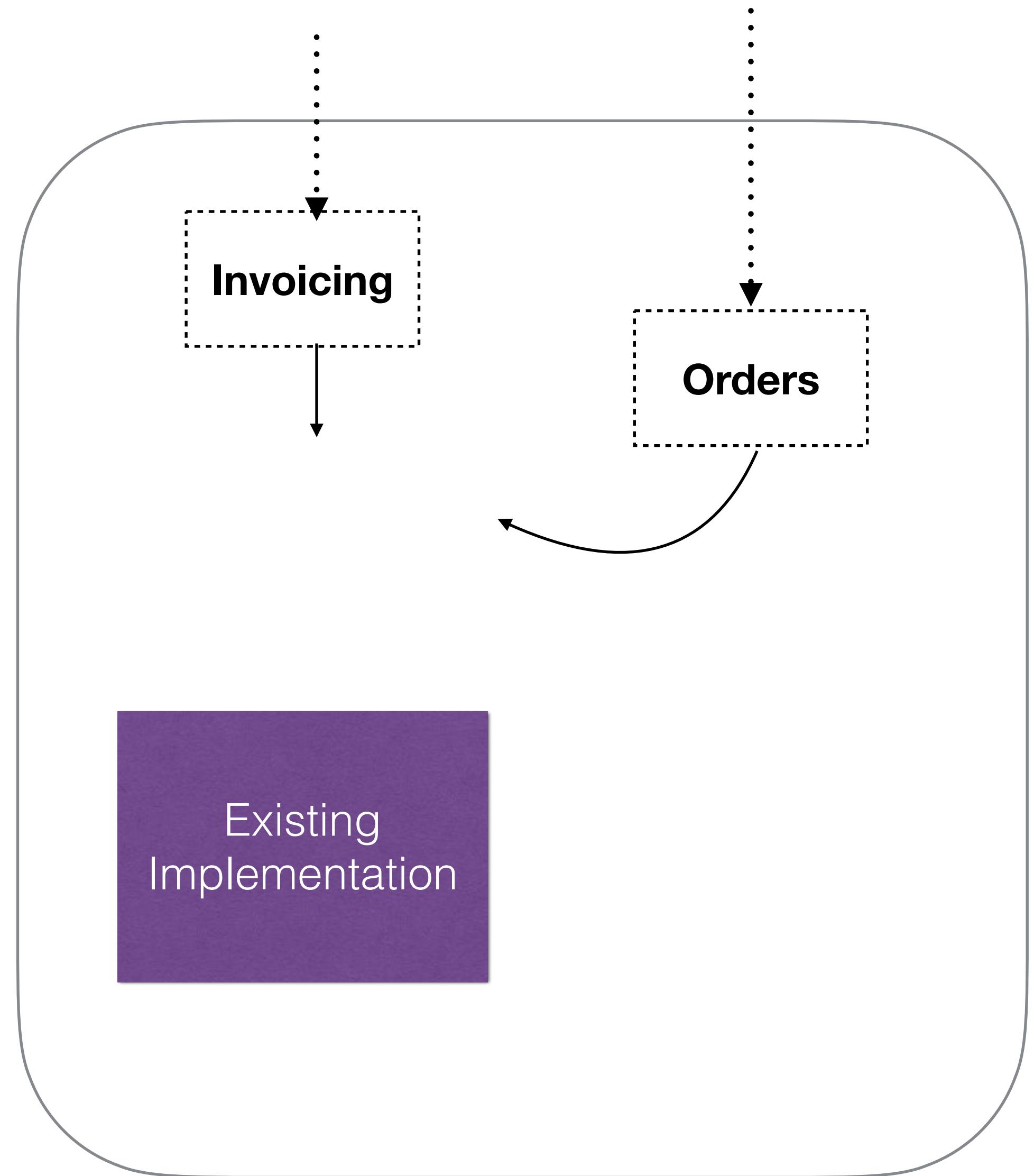
## EXAMPLE



## EXAMPLE

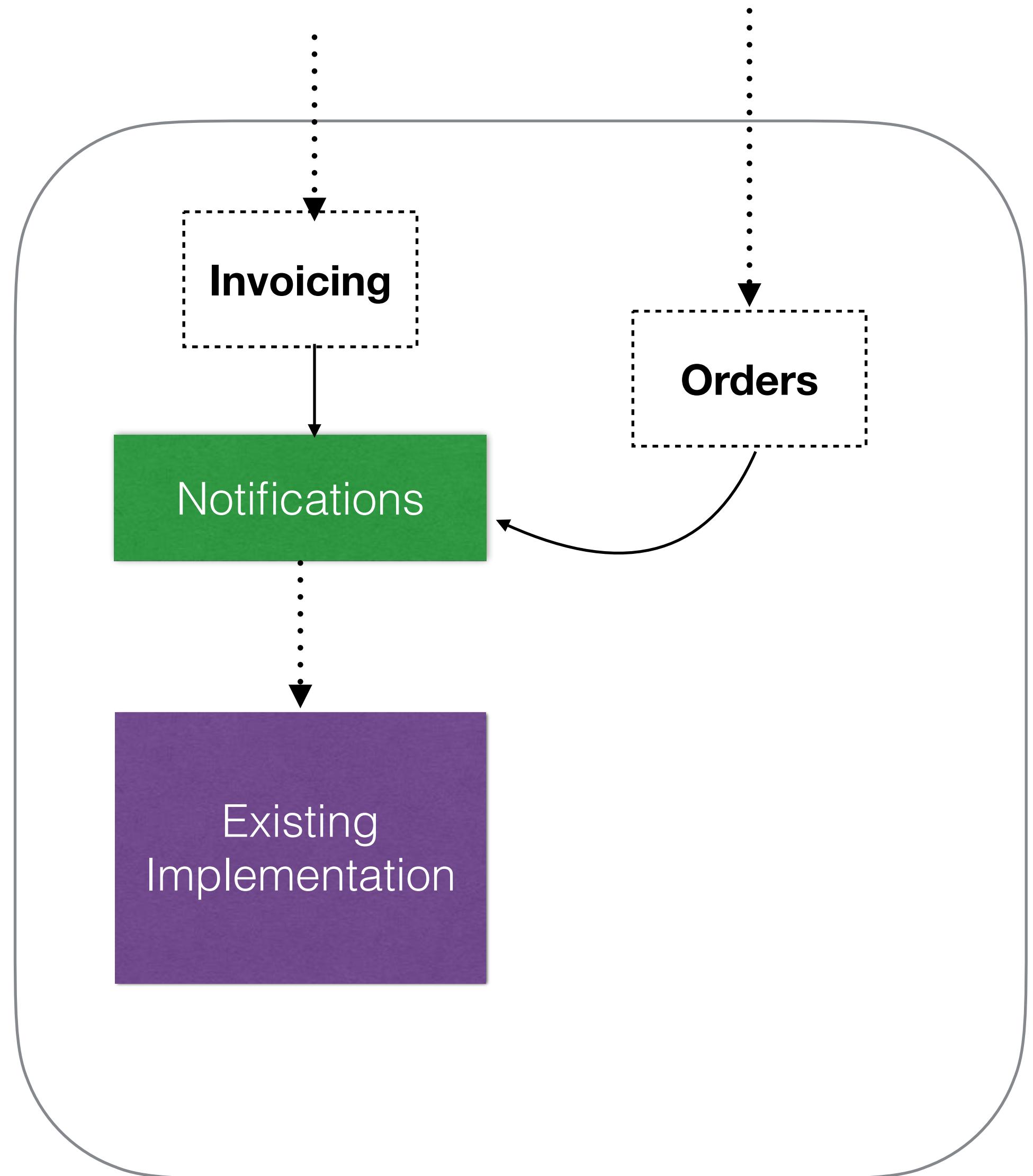


## EXAMPLE

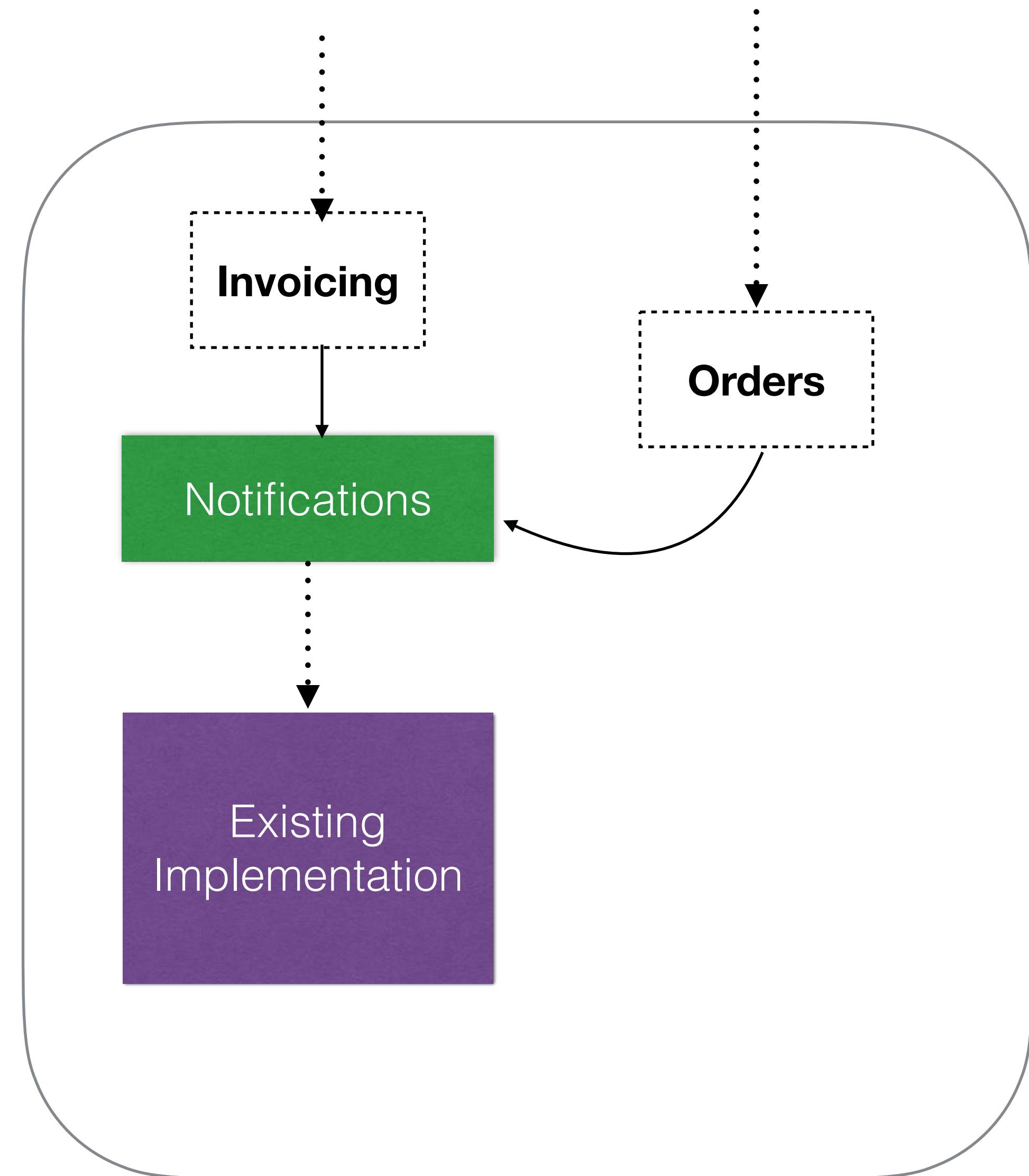


## EXAMPLE

### 1. Create abstraction point



## EXAMPLE

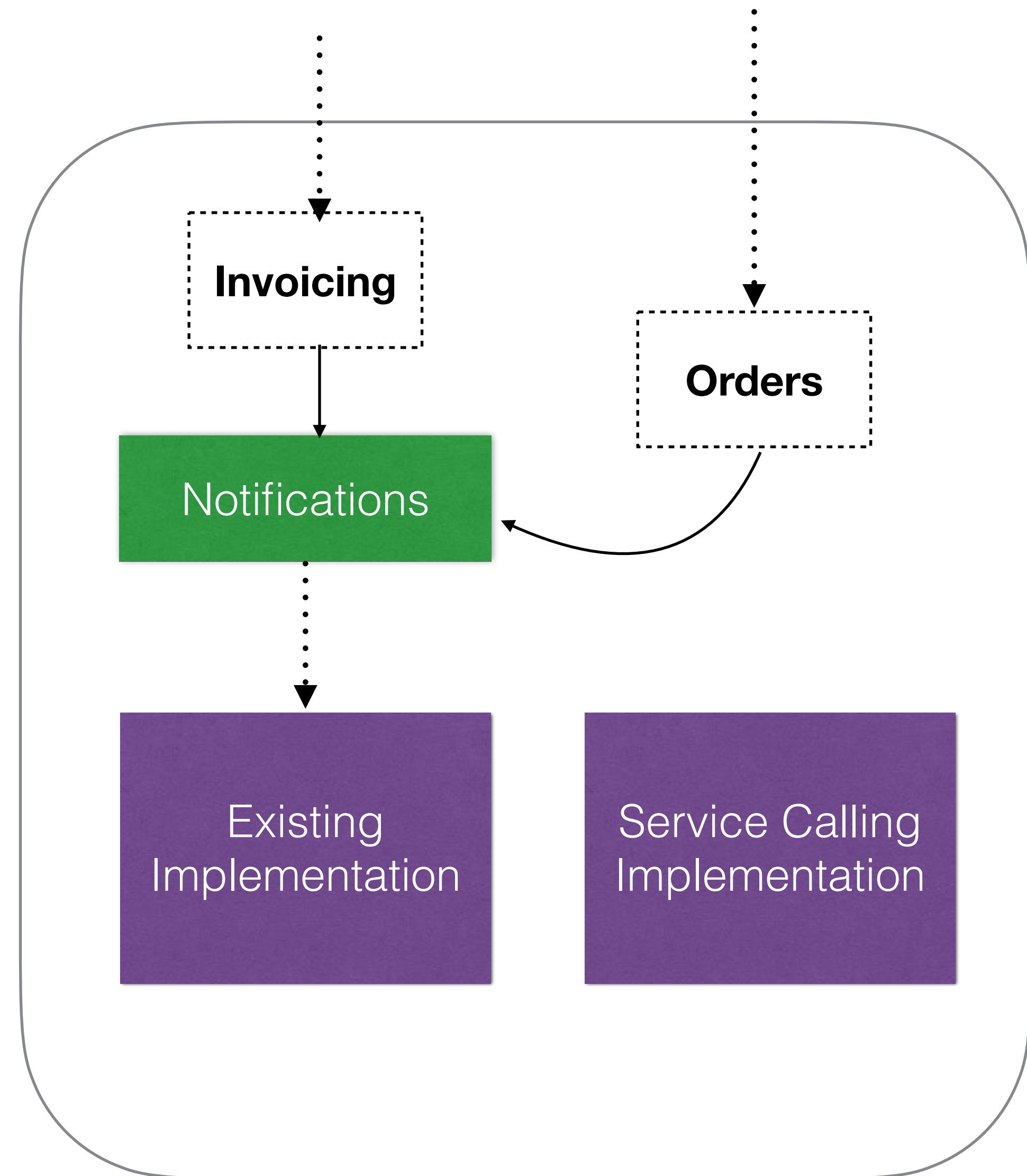


1. Create abstraction point
2. Start work on new service implementation

## EXAMPLE

1. Create abstraction point

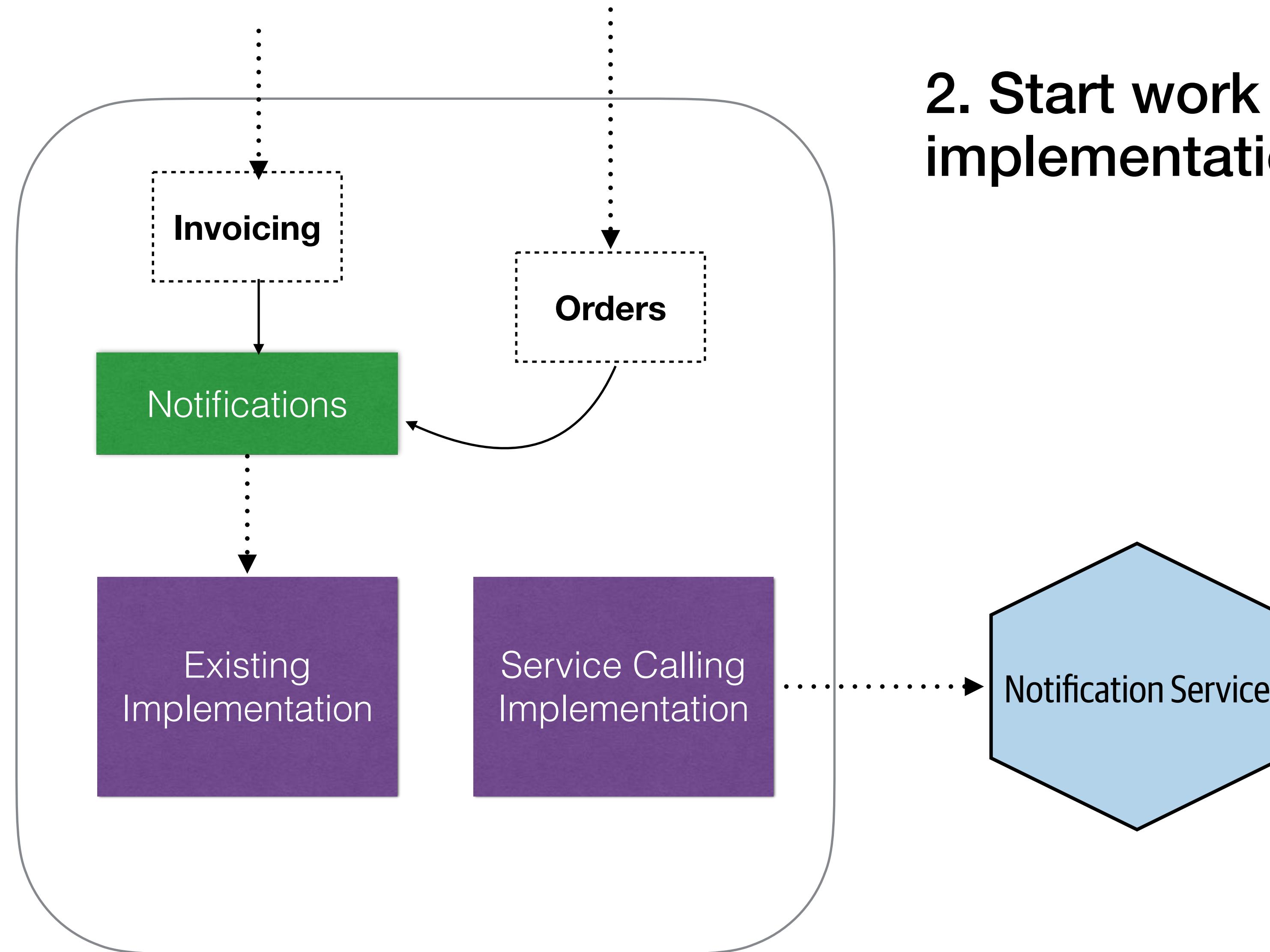
2. Start work on new service implementation



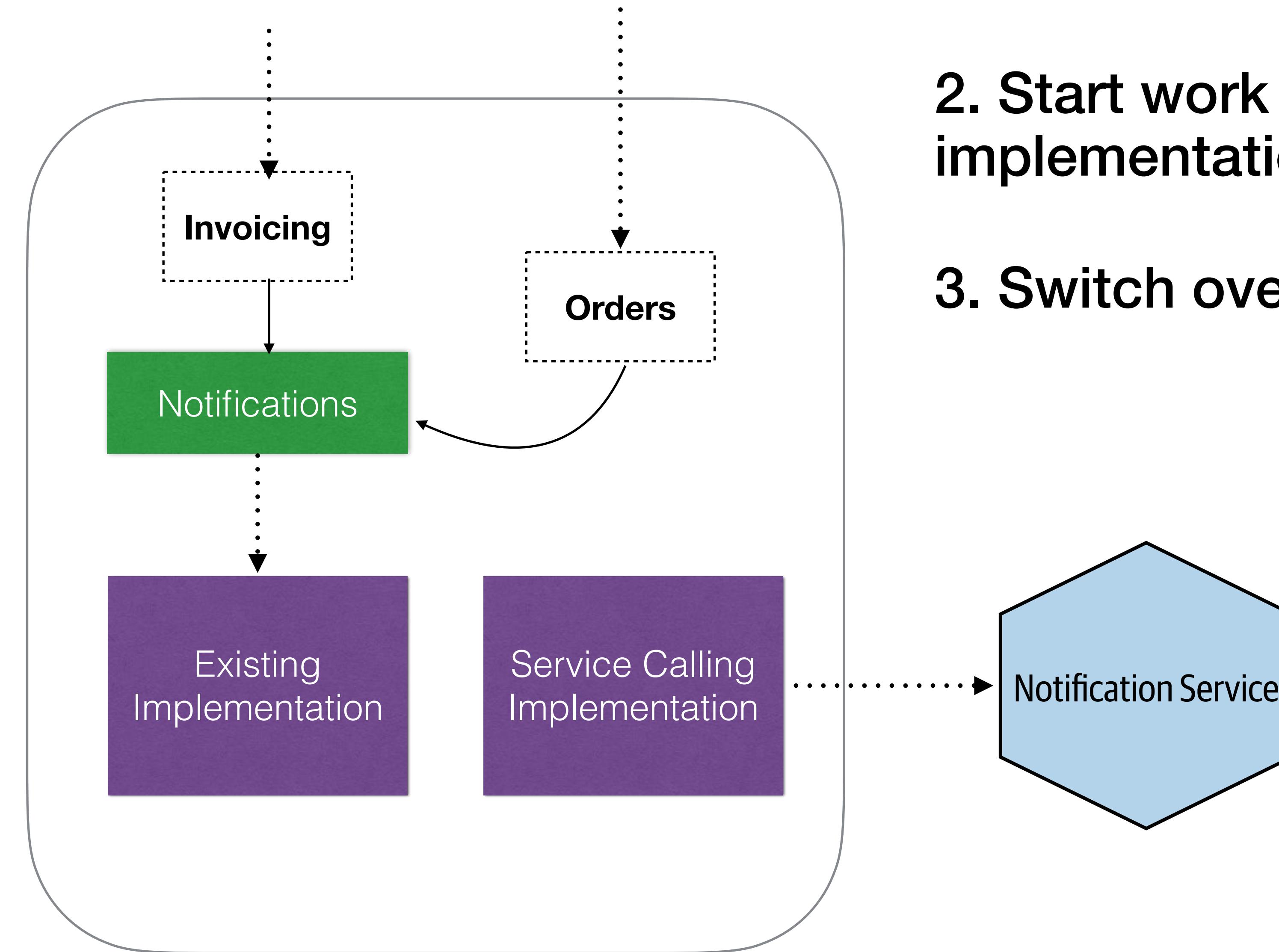
## EXAMPLE

1. Create abstraction point

2. Start work on new service implementation

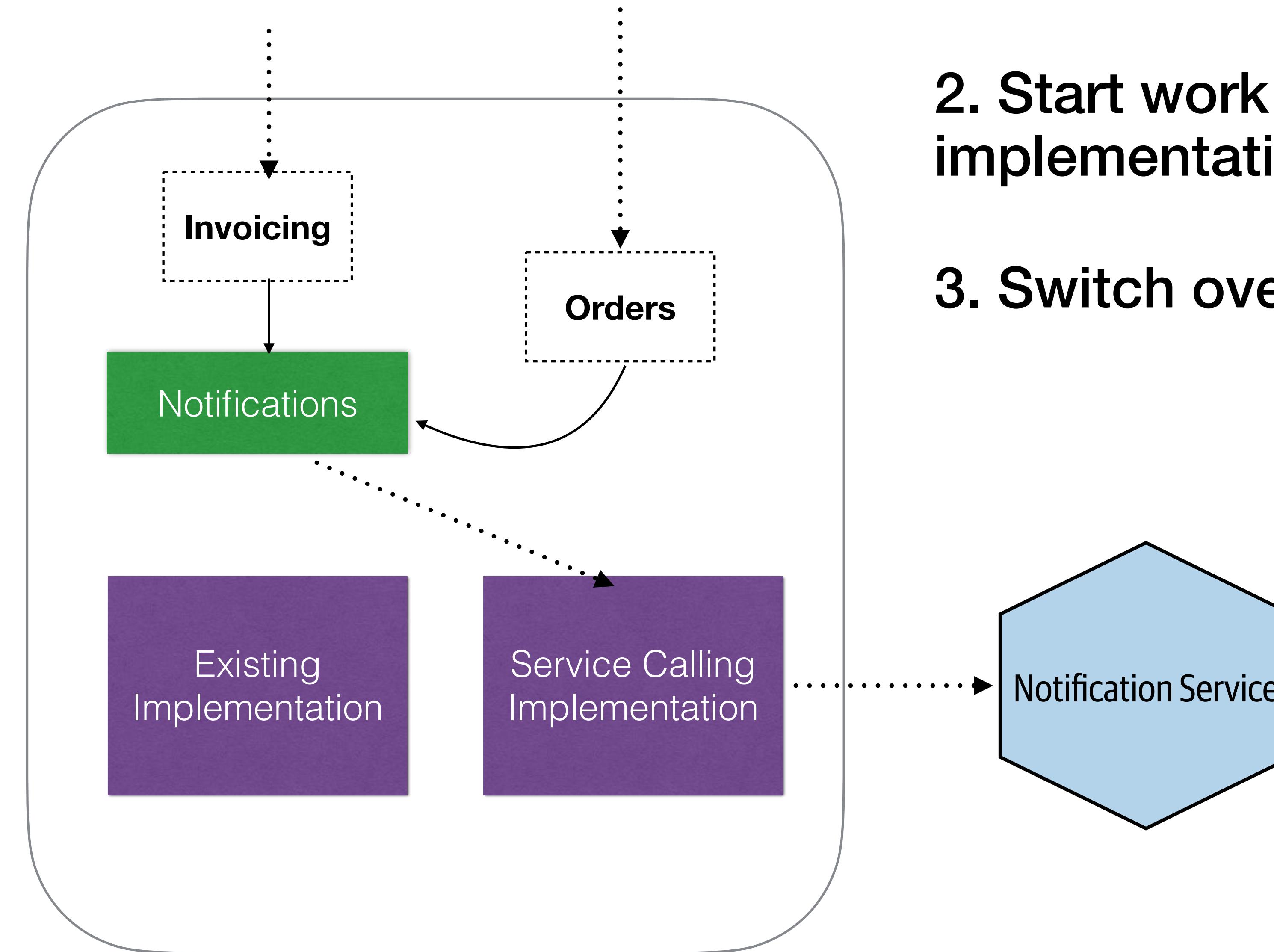


## EXAMPLE



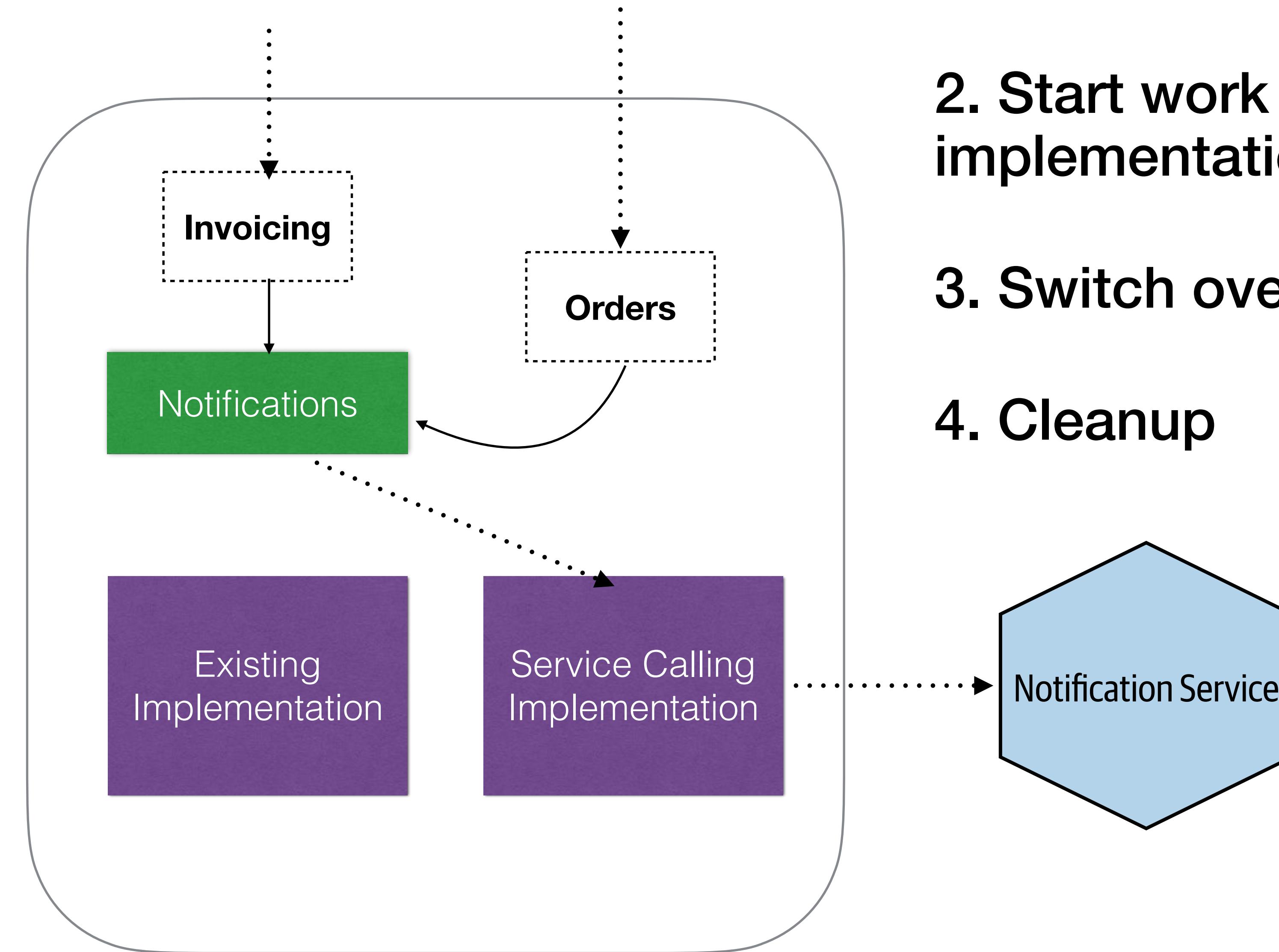
1. Create abstraction point
2. Start work on new service implementation
3. Switch over

## EXAMPLE



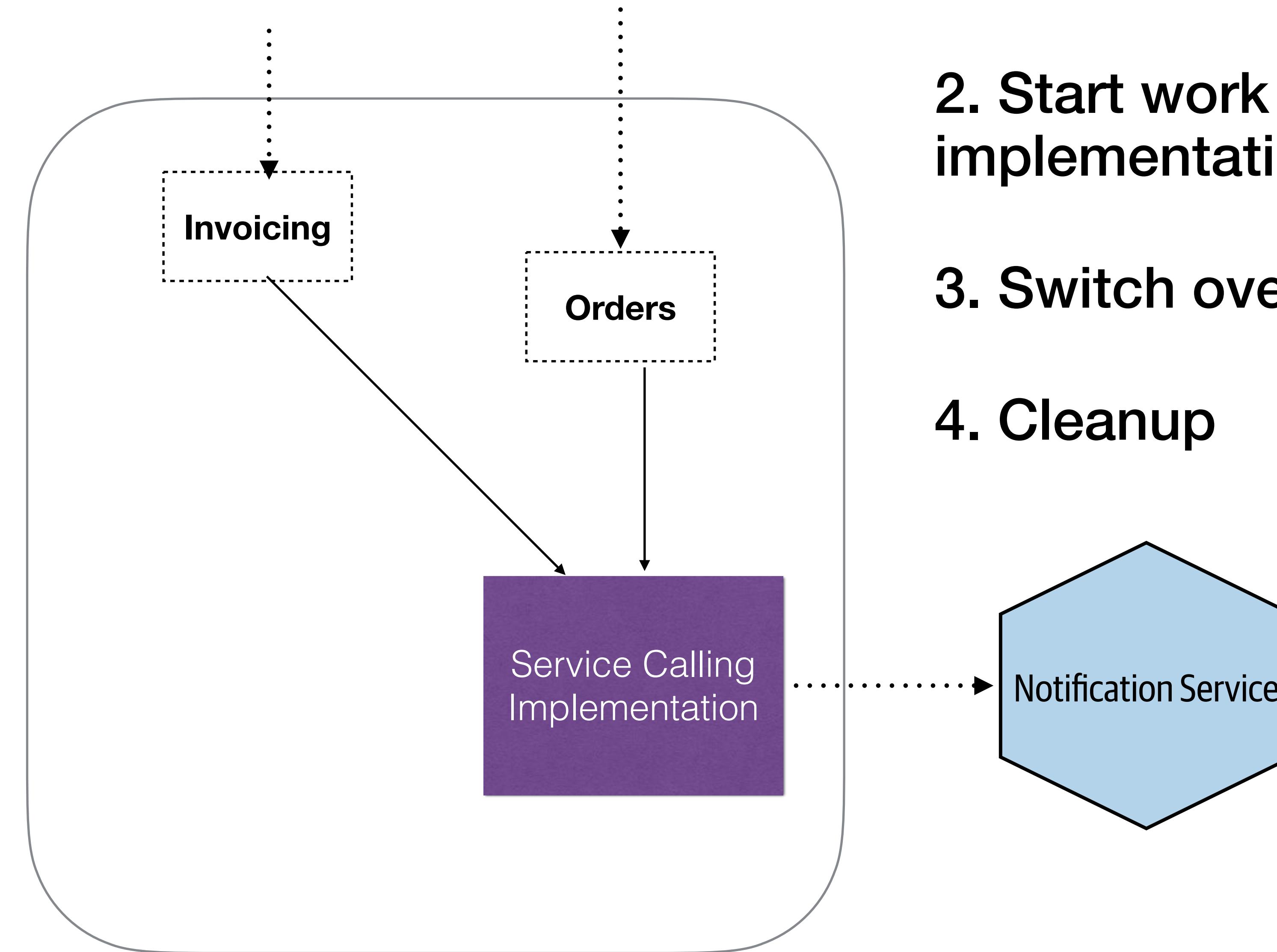
1. Create abstraction point
2. Start work on new service implementation
3. Switch over

## EXAMPLE

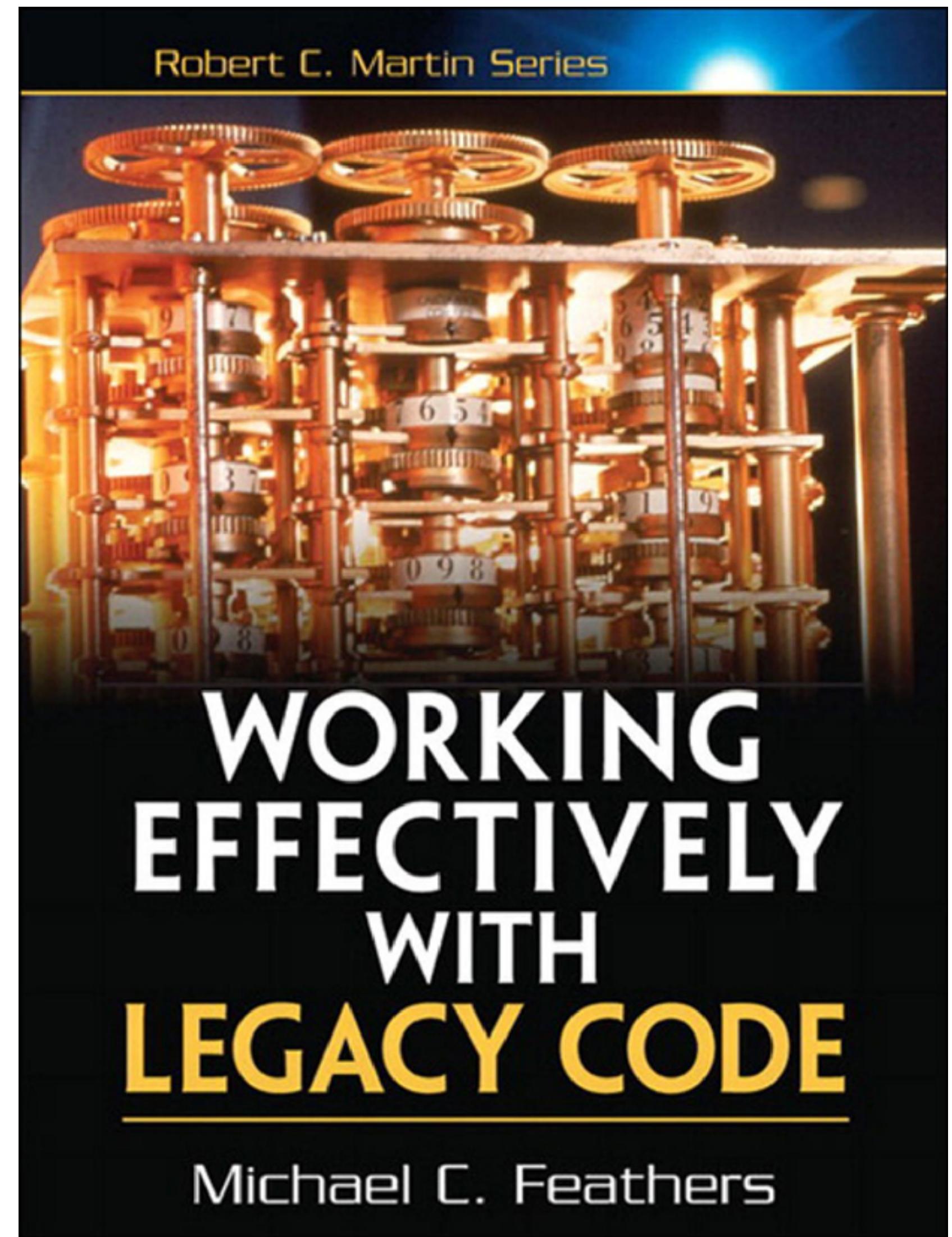


- 1. Create abstraction point**
- 2. Start work on new service implementation**
- 3. Switch over**
- 4. Cleanup**

## EXAMPLE



1. Create abstraction point
2. Start work on new service implementation
3. Switch over
4. Cleanup



## PATTERN: FEATURE TOGGLES



# PATTERN: FEATURE TOGGLES

## Feature Toggles (aka Feature Flags)

Feature Toggles (often also referred to as Feature Flags) are a powerful technique, allowing teams to modify system behavior without changing code. They fall into various usage categories, and it's important to take that categorization into account when implementing and managing toggles. Toggles introduce complexity. We can keep that complexity in check by using smart toggle implementation practices and appropriate tools to manage our toggle configuration, but we should also aim to constrain the number of toggles in our system.

09 October 2017



### CONTENTS

[expand](#)

- [A Toggling Tale](#)
- [Categories of toggles](#)
- [Implementation Techniques](#)
- [Toggle Configuration](#)
- [Working with feature-flagged systems](#)

Pete Hodgson

Pete Hodgson is an independent  
software delivery consultant based in  
the San Francisco Bay Area. He  
specializes in helping startup

## Allow for functionality to be toggled off and on

# PATTERN: FEATURE TOGGLES

## Feature Toggles (aka Feature Flags)

Feature Toggles (often also referred to as Feature Flags) are a powerful technique, allowing teams to modify system behavior without changing code. They fall into various usage categories, and it's important to take that categorization into account when implementing and managing toggles. Toggles introduce complexity. We can keep that complexity in check by using smart toggle implementation practices and appropriate tools to manage our toggle configuration, but we should also aim to constrain the number of toggles in our system.

09 October 2017



### CONTENTS

[expand](#)

- [A Toggling Tale](#)
- [Categories of toggles](#)
- [Implementation Techniques](#)
- [Toggle Configuration](#)
- [Working with feature-flagged systems](#)

Pete Hodgson

Pete Hodgson is an independent  
software delivery consultant based in  
the San Francisco Bay Area. He  
specializes in helping startup

**Allow for functionality to be toggled off and on**

**Can also be used to switch between alternative implementations**

# PATTERN: FEATURE TOGGLES

## Feature Toggles (aka Feature Flags)

Feature Toggles (often also referred to as Feature Flags) are a powerful technique, allowing teams to modify system behavior without changing code. They fall into various usage categories, and it's important to take that categorization into account when implementing and managing toggles. Toggles introduce complexity. We can keep that complexity in check by using smart toggle implementation practices and appropriate tools to manage our toggle configuration, but we should also aim to constrain the number of toggles in our system.

09 October 2017



Pete Hodgson

Pete Hodgson is an independent software delivery consultant based in the San Francisco Bay Area. He specializes in helping startup

### CONTENTS

[expand](#)

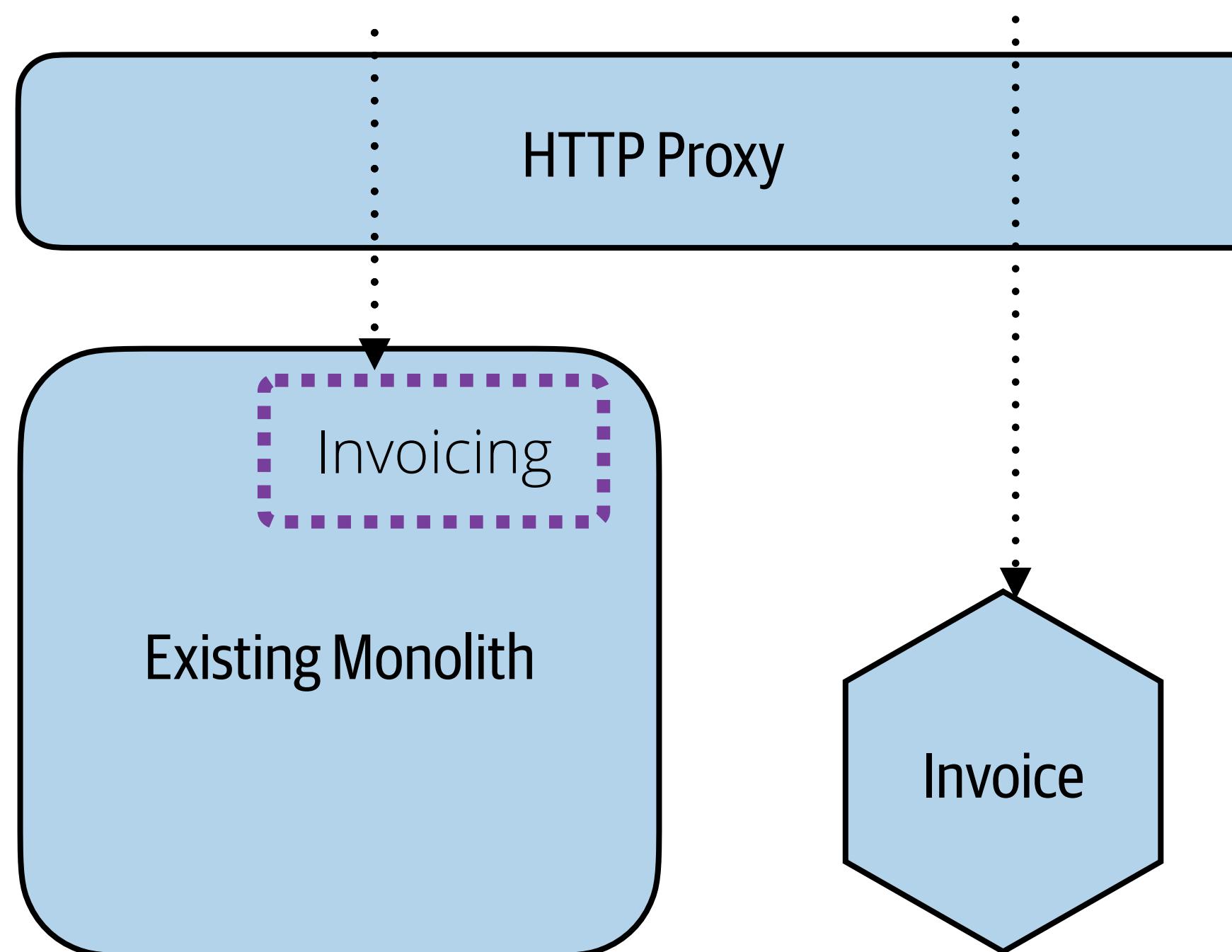
- [A Toggling Tale](#)
- [Categories of toggles](#)
- [Implementation Techniques](#)
- [Toggle Configuration](#)
- [Working with feature-flagged systems](#)

**Allow for functionality to be toggled off and on**

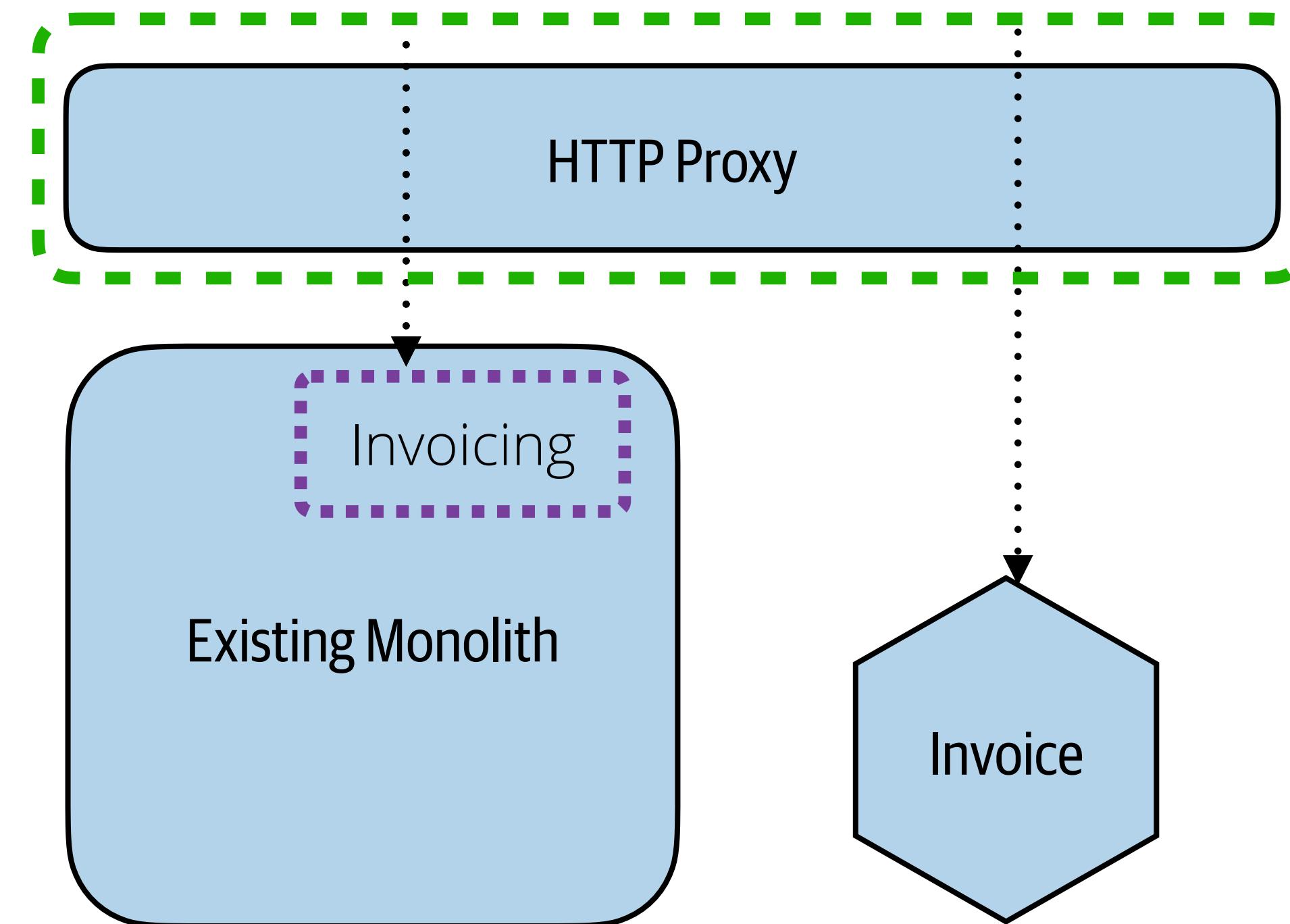
**Can also be used to switch between alternative implementations**

**Changing toggled values doesn't require a rebuild or redeploy**

## WITH THE STRANGLER FIG PATTERN

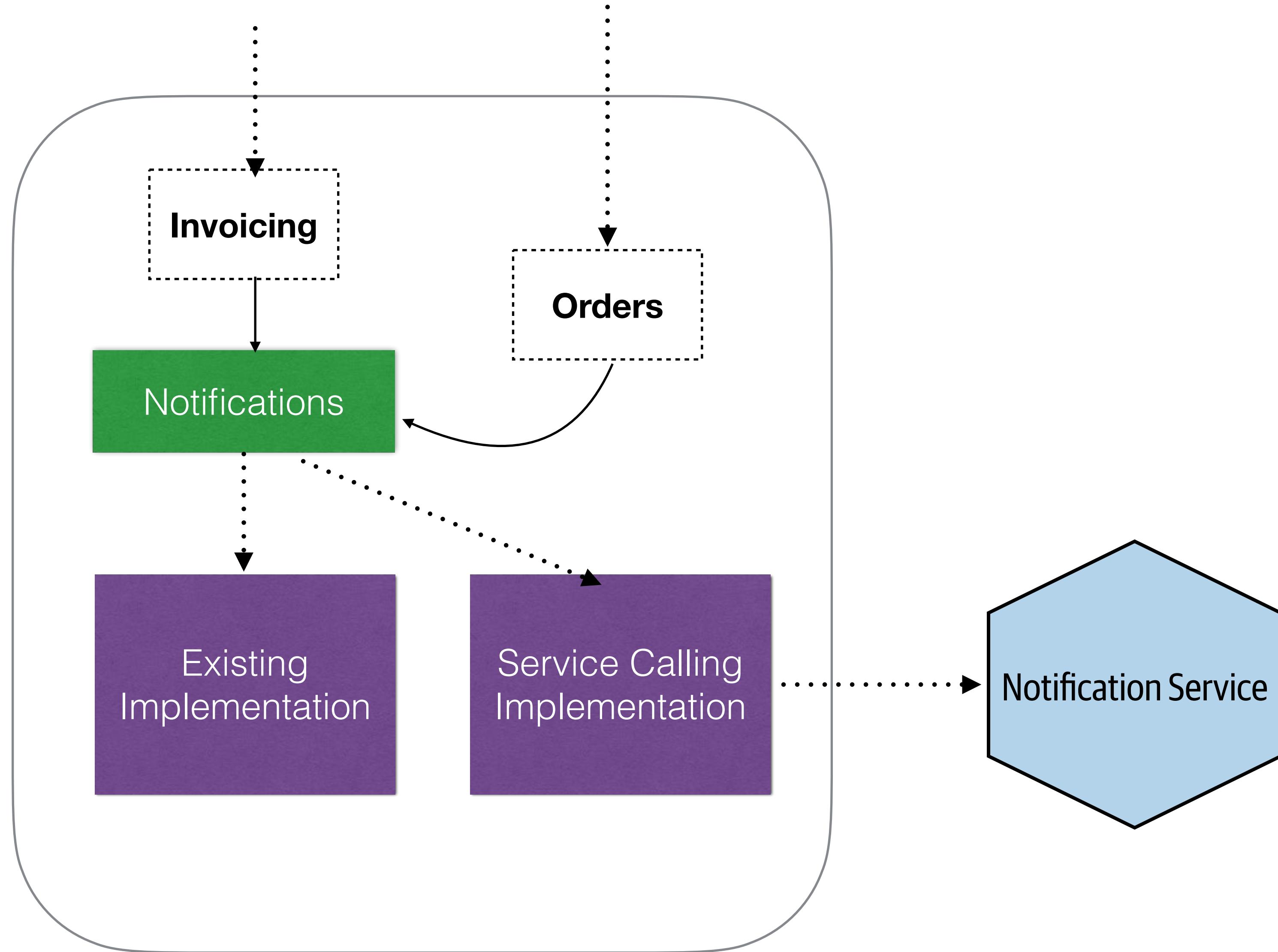


## WITH THE STRANGLER FIG PATTERN



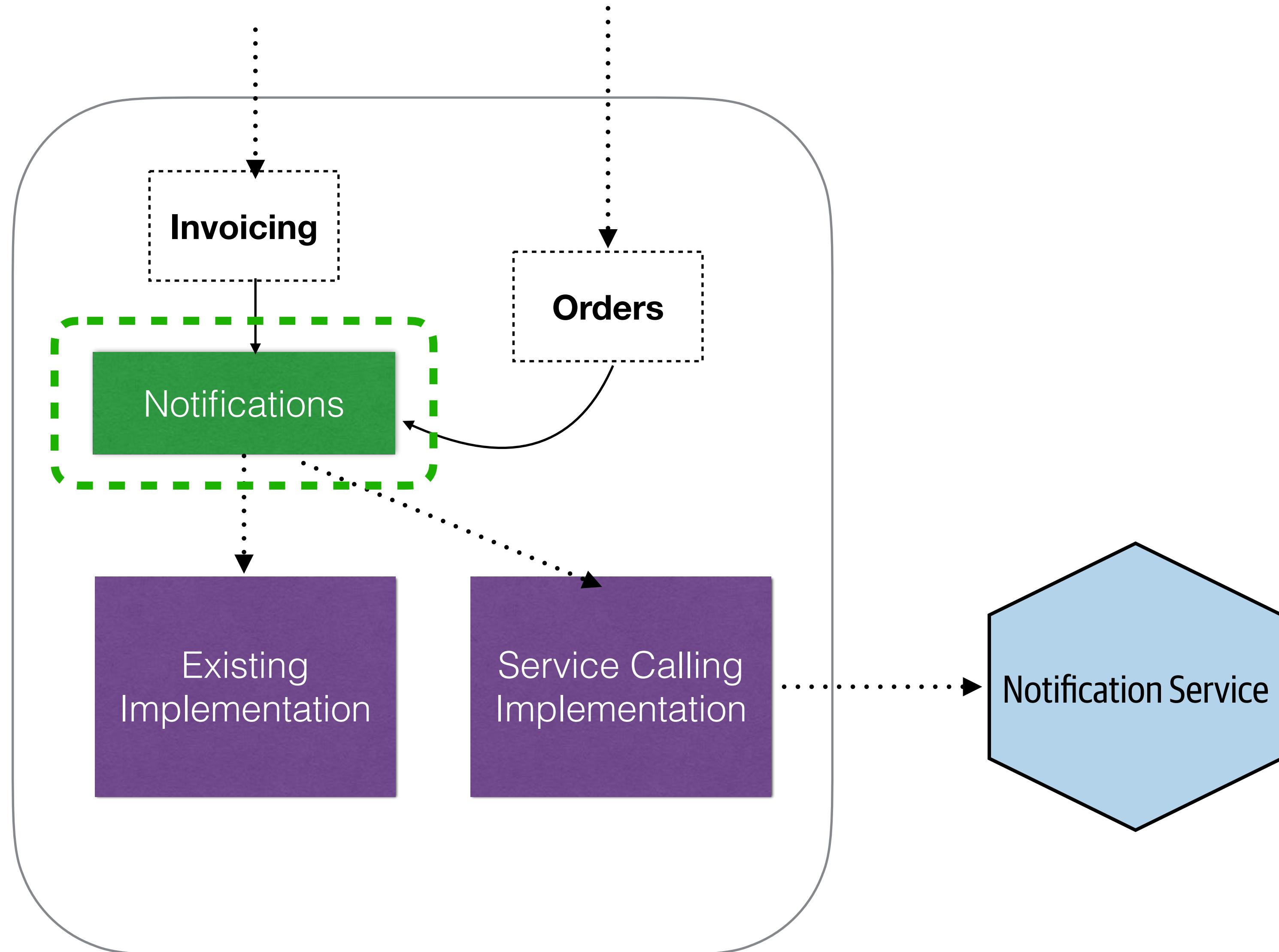
**Implement a simple config-based toggle in the HTTP Proxy configuration**

## WITH BRANCH BY ABSTRACTION

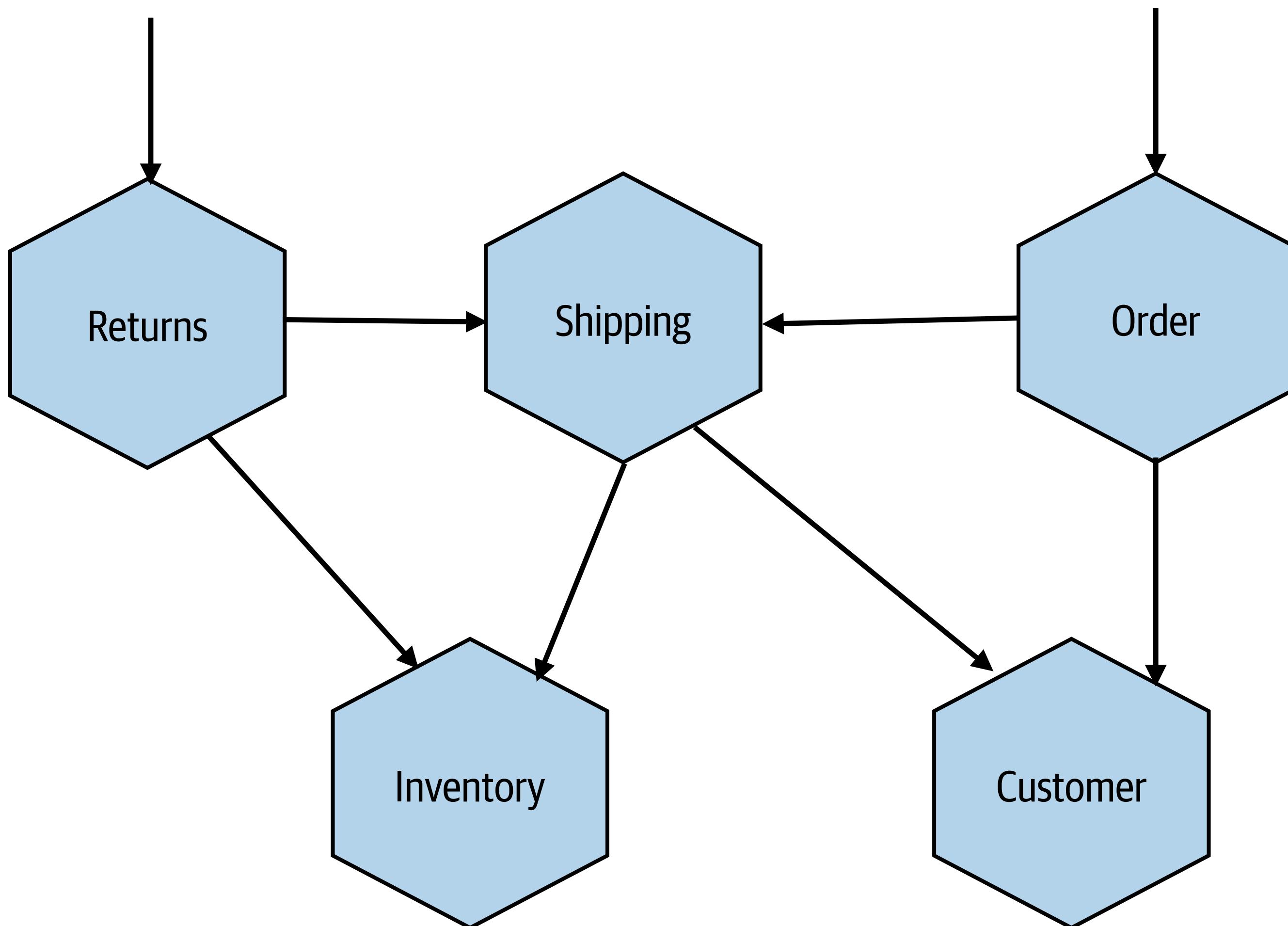


## WITH BRANCH BY ABSTRACTION

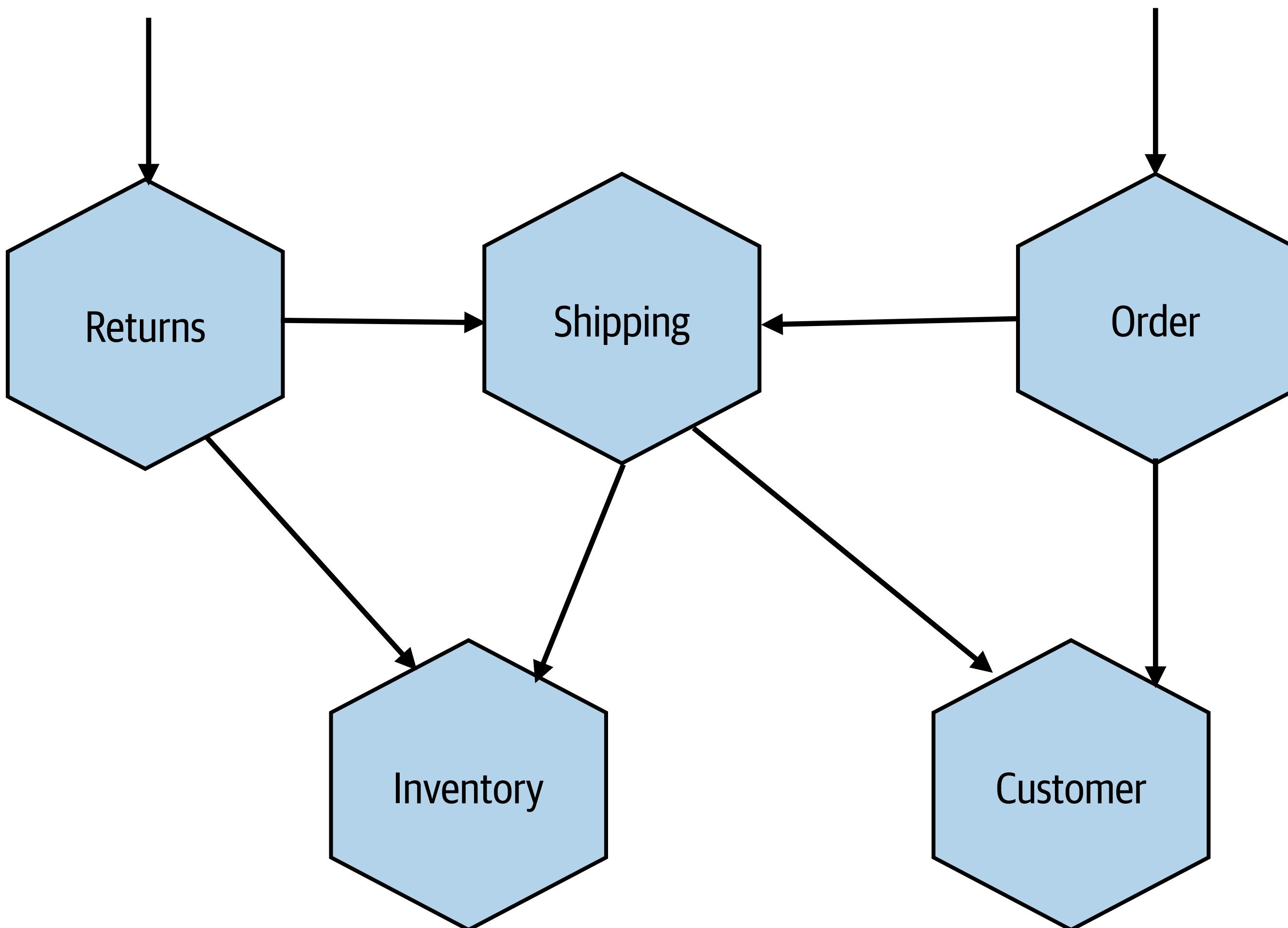
Use the abstraction to divert calls based on toggle value



## BRIEF RECAP...

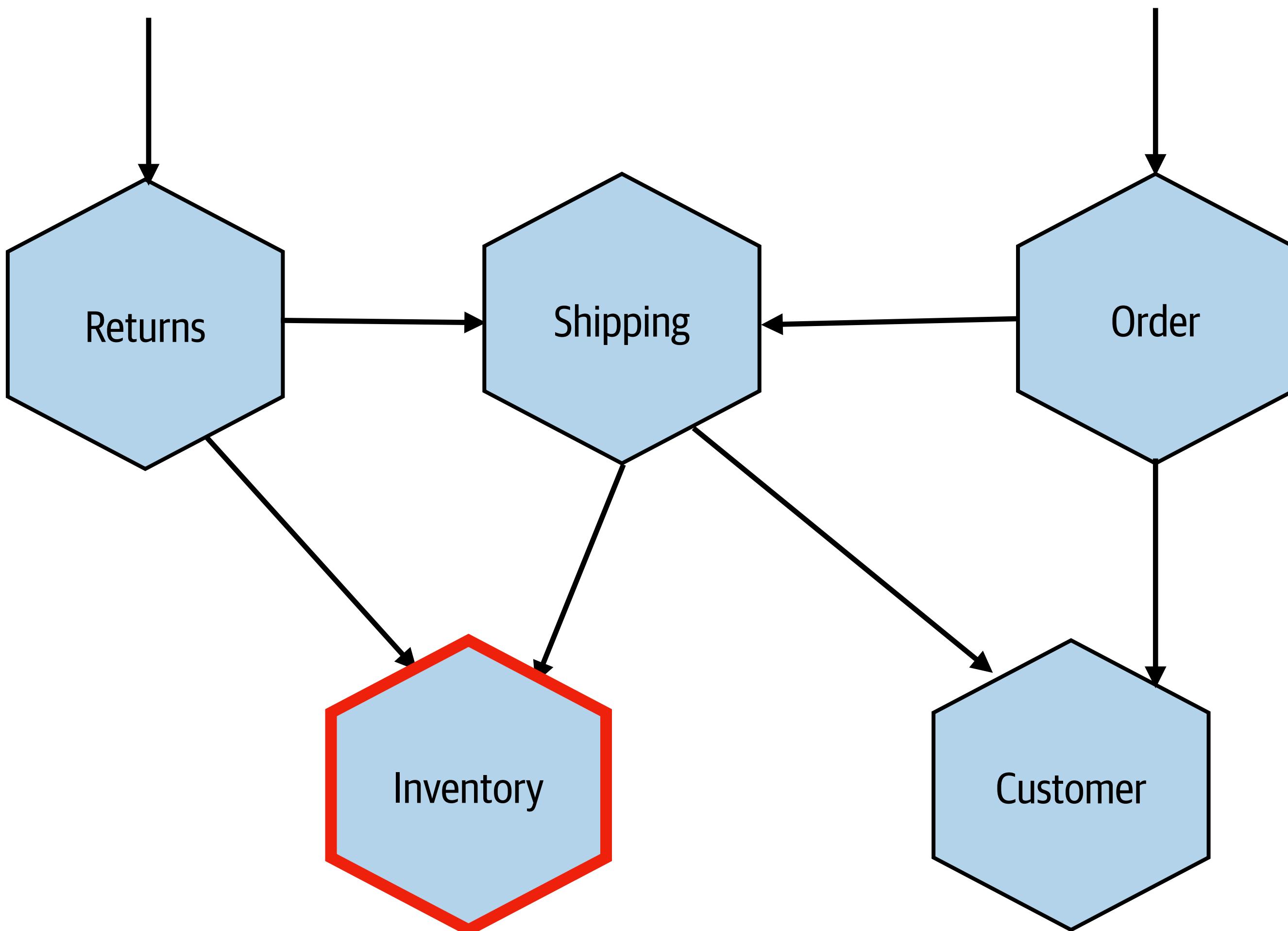


## BRIEF RECAP...



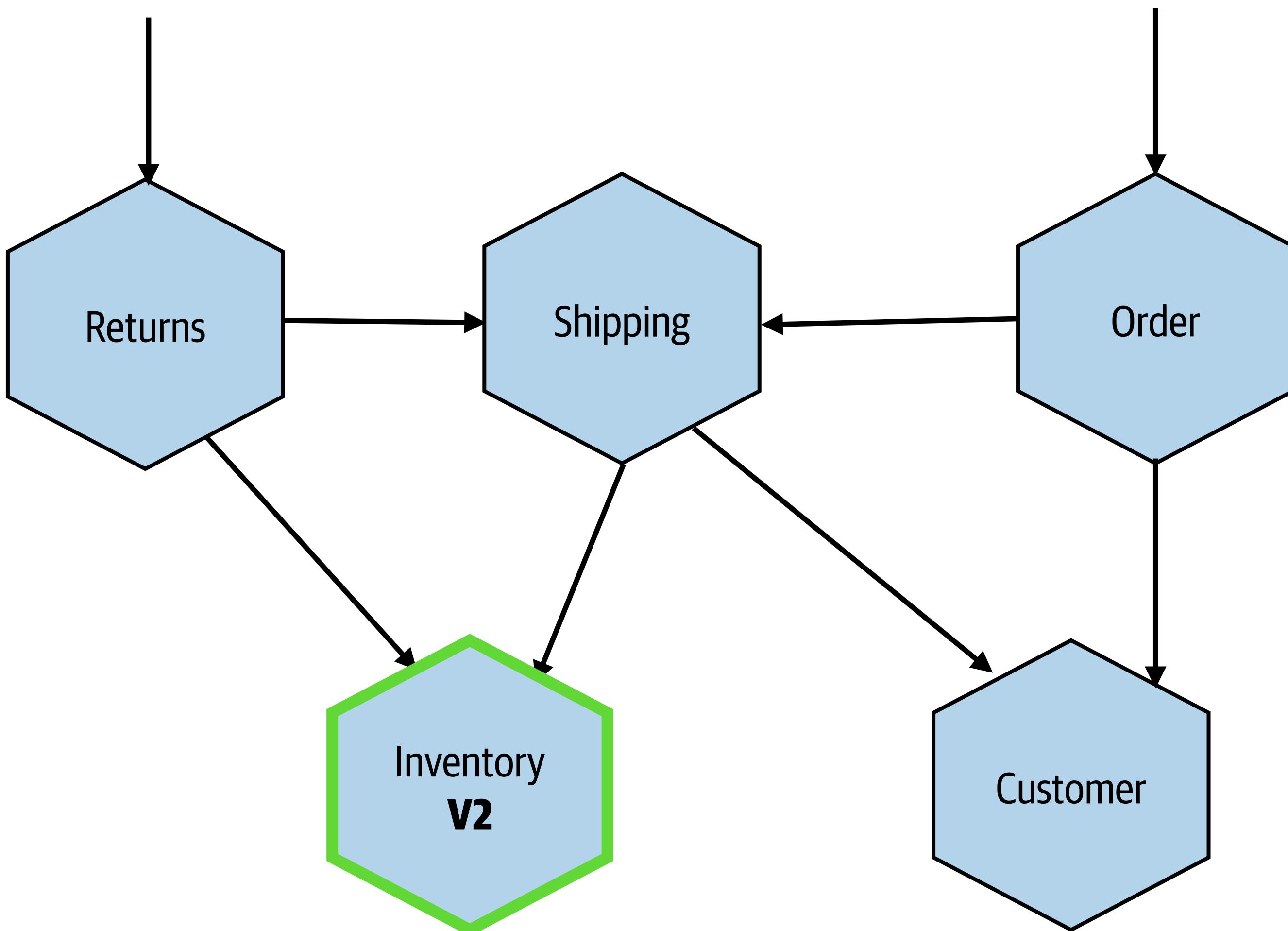
Independently  
Deployable

## BRIEF RECAP...



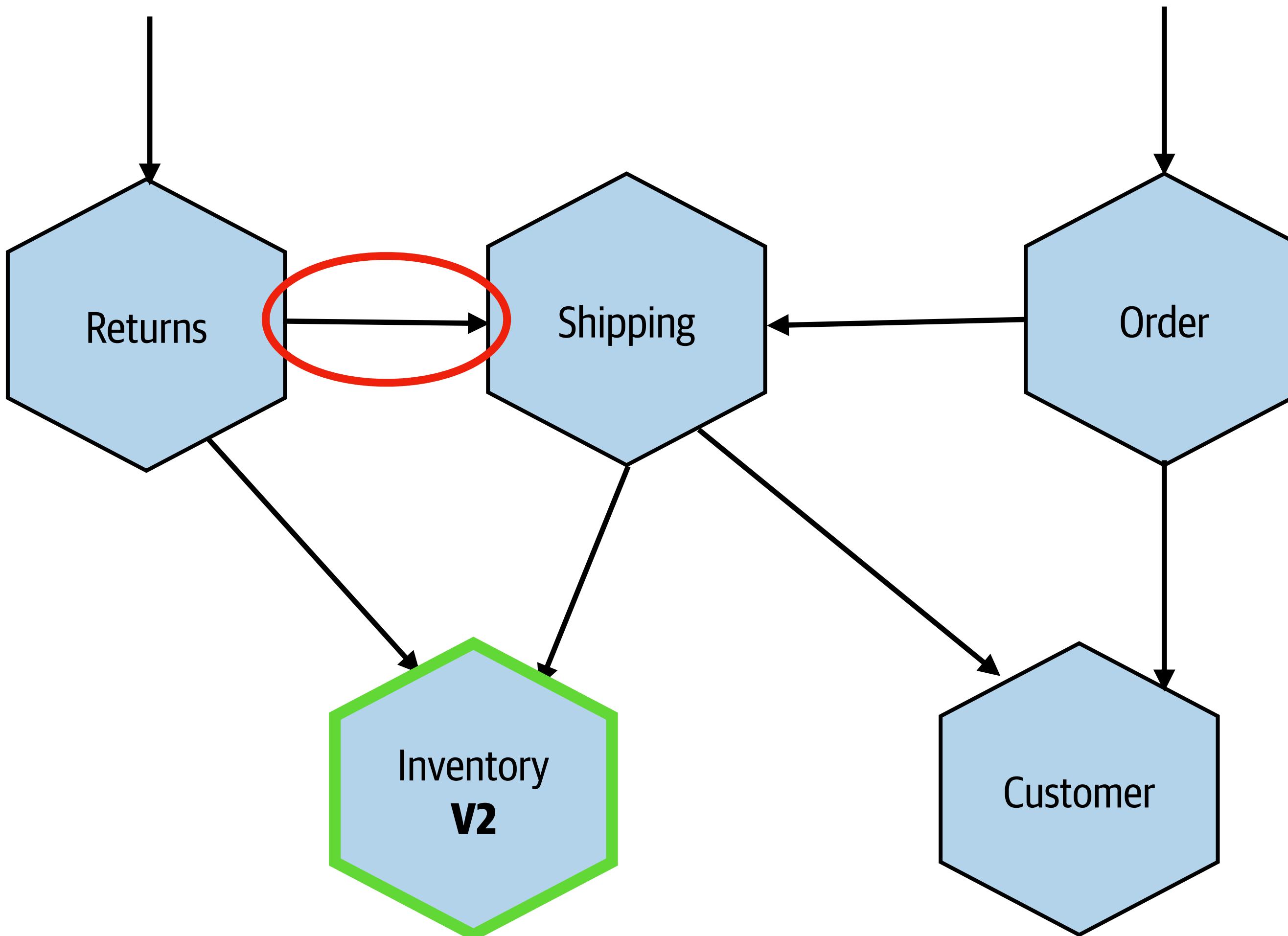
Independently  
Deployable

## BRIEF RECAP...



Independently  
Deployable

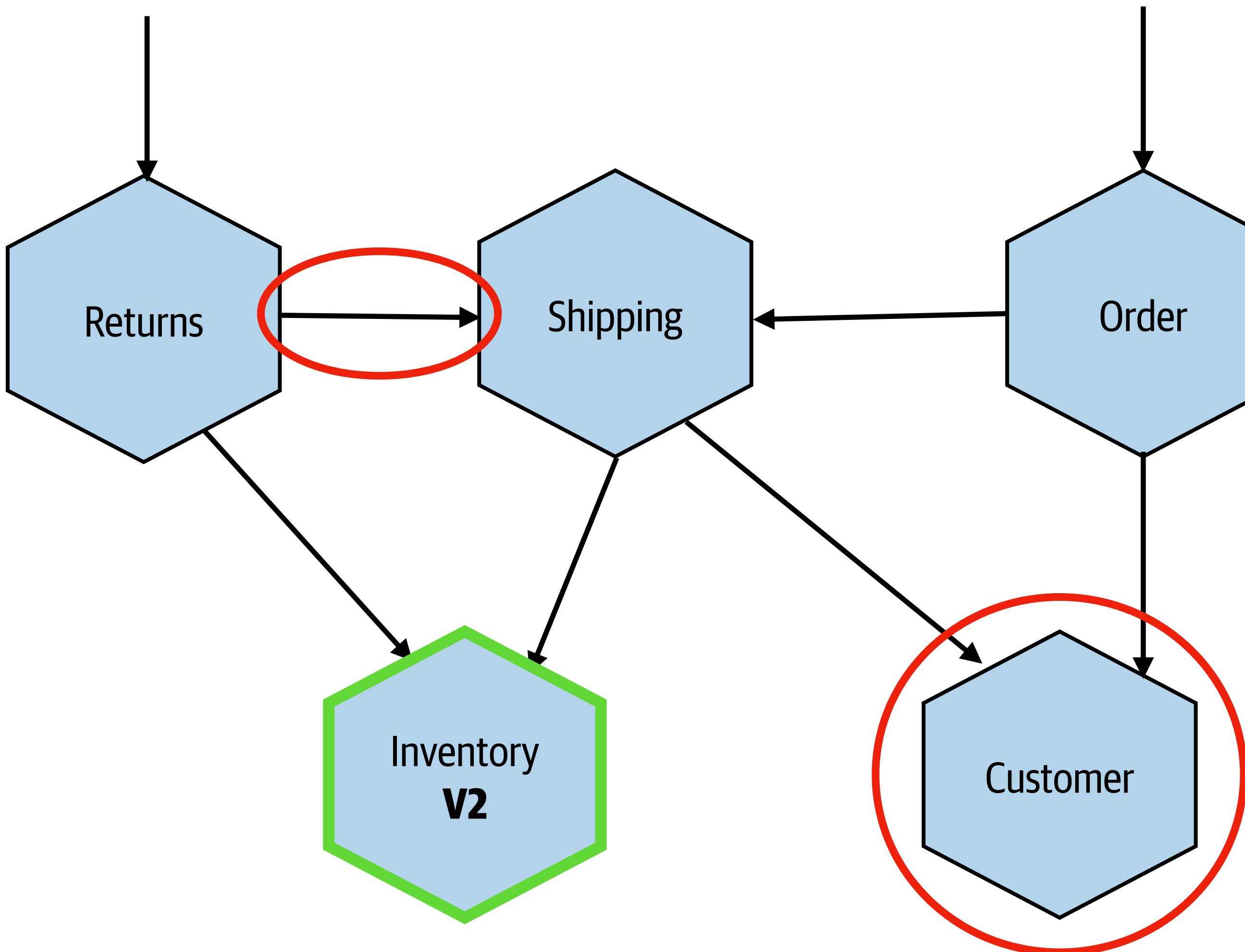
## BRIEF RECAP...



Independently Deployable

Microservices often depend on other microservices

## BRIEF RECAP...



**Independently Deployable**

**Microservices often depend on other microservices**

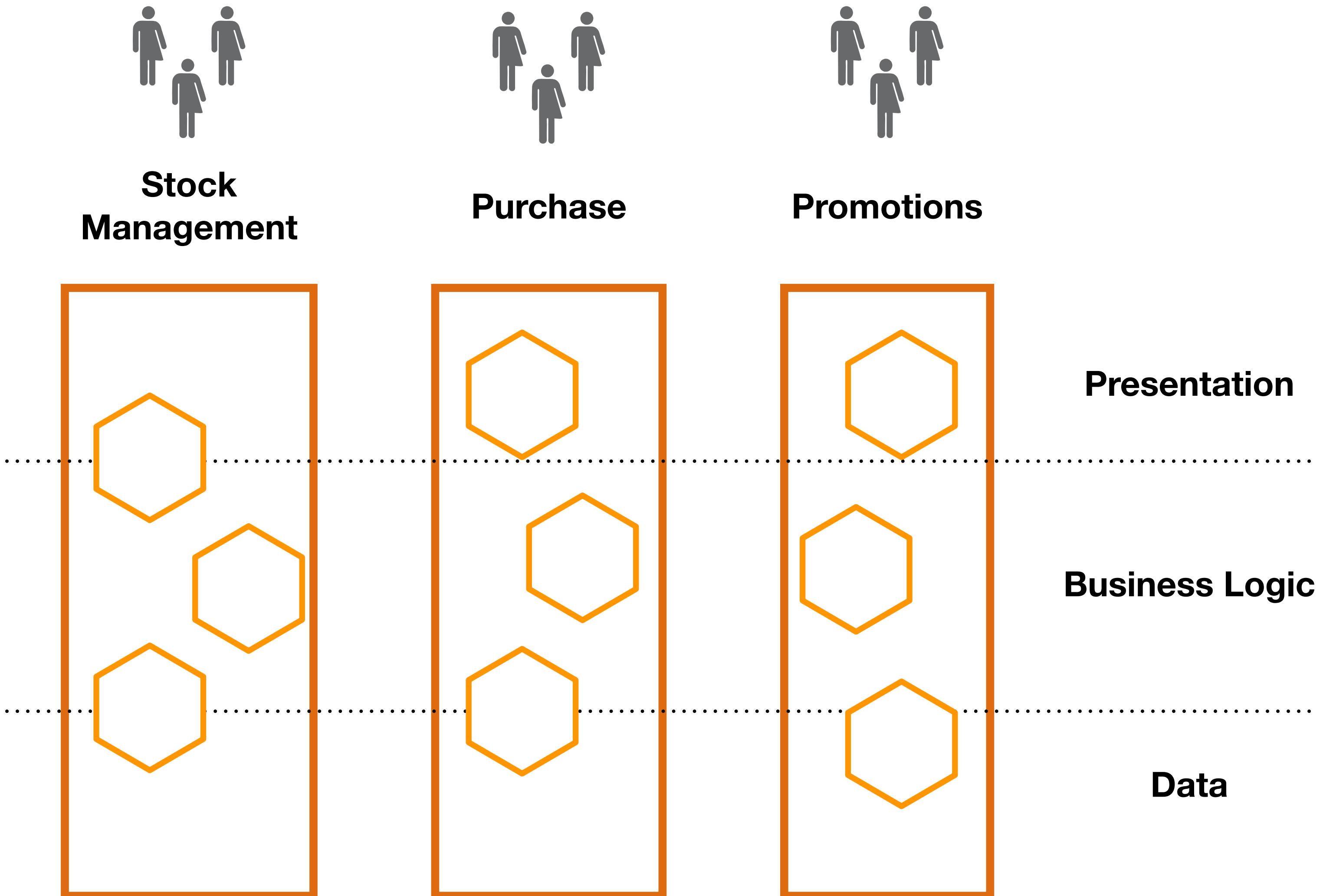
**Boundaries primarily defined by the domain**

## STREAM-ALIGNED TEAMS

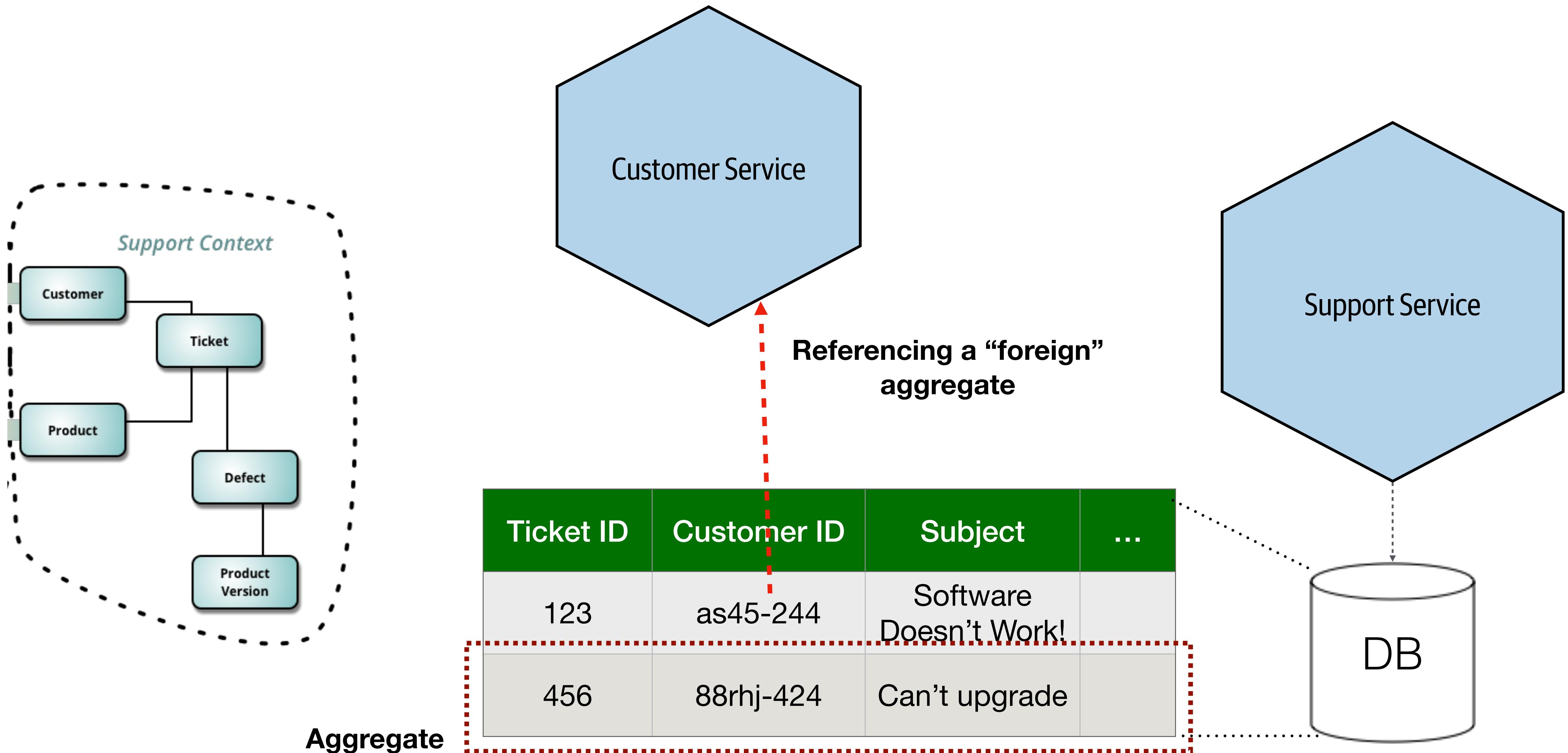
Focused on a valuable stream of work

Long-lived “product” oriented rather than project oriented

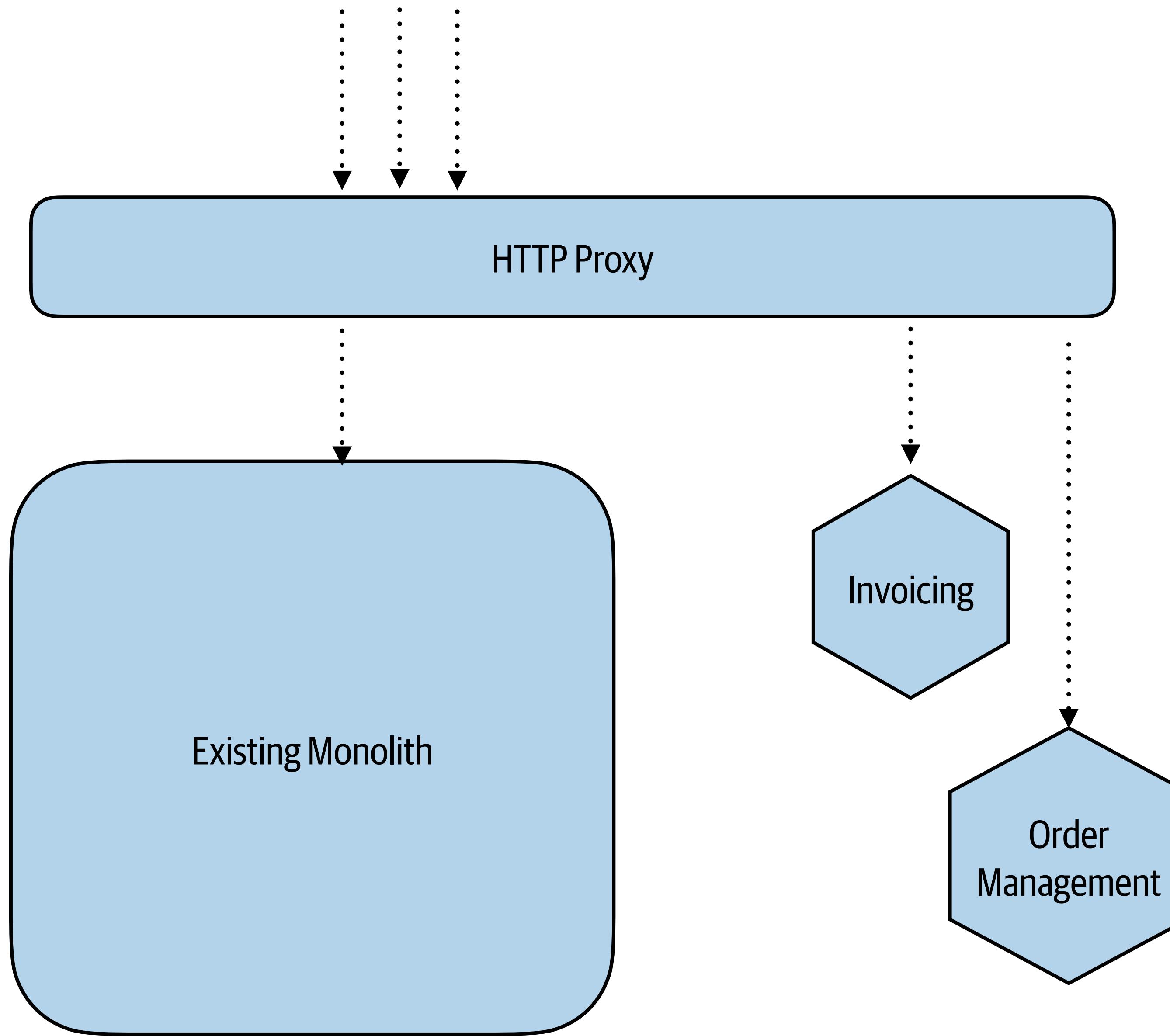
Has ownership of their assets



## IMPLEMENTATION EXAMPLE...



# STRANGLER FIG



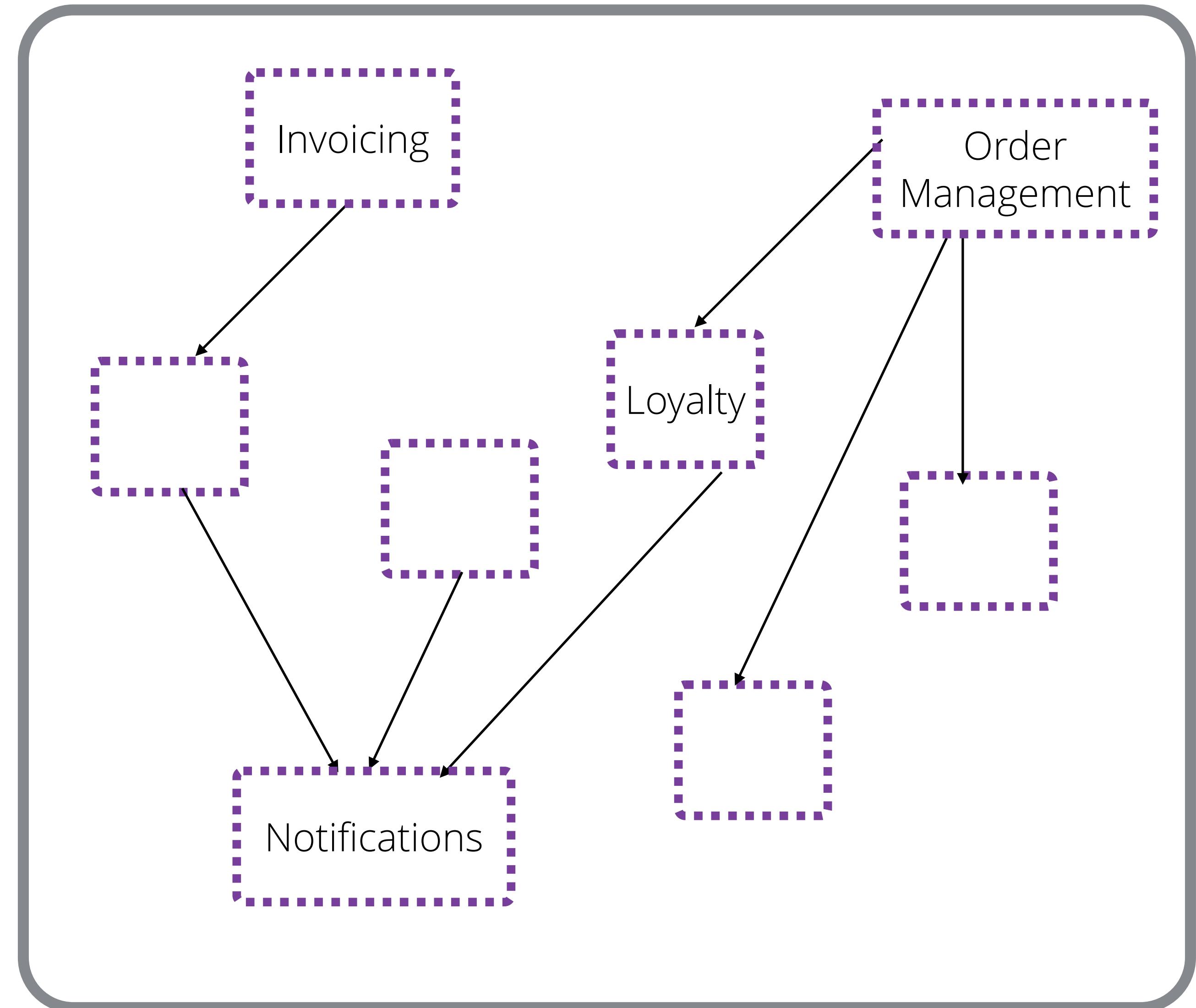
# Questions from last week?

**Picking up where we left off...**

# **2/4 App Decomposition**

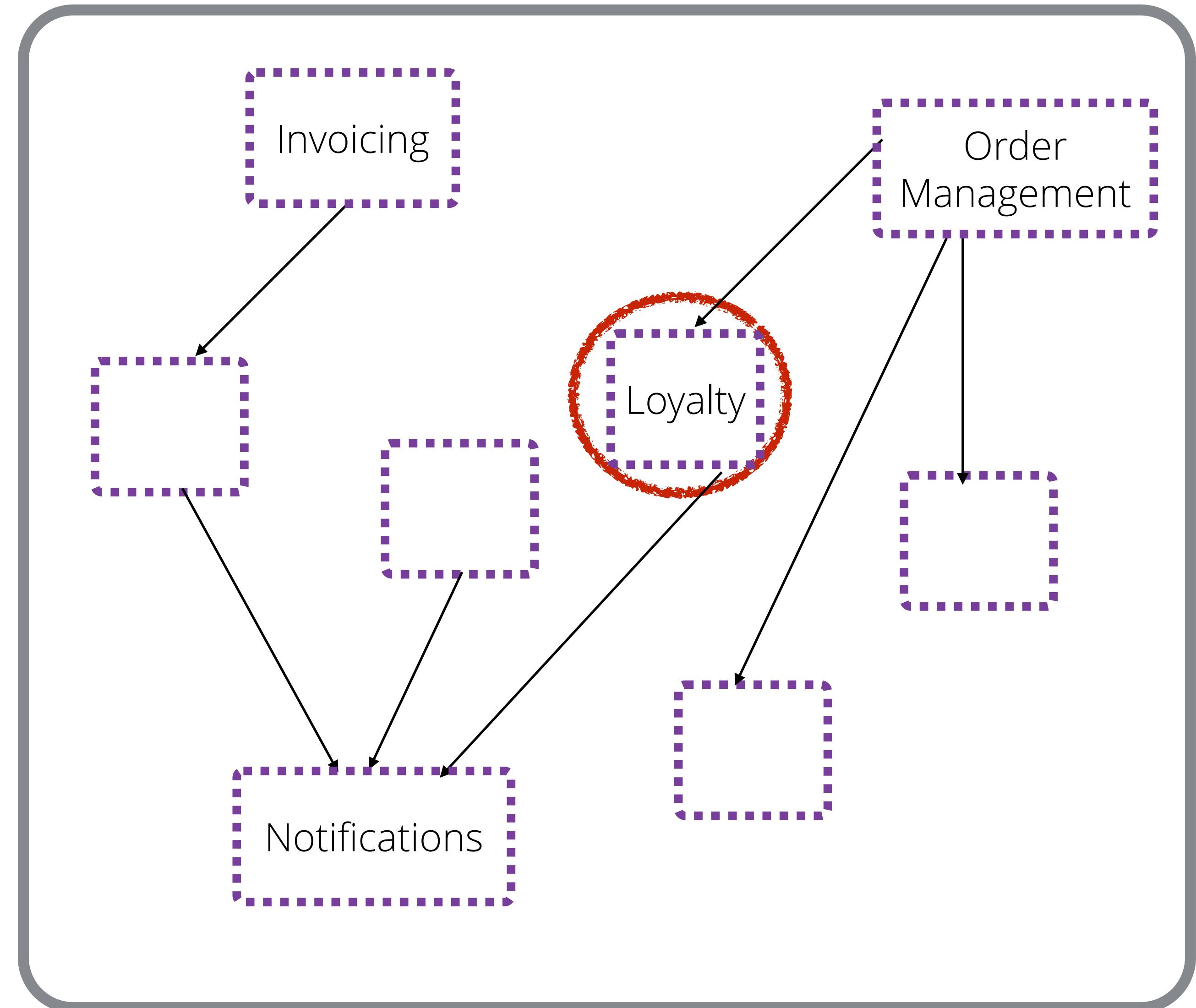
## WHAT ABOUT OTHER FUNCTIONALITY?

Can use “branch by abstraction” to help extraction of functionality deeper in the monolith



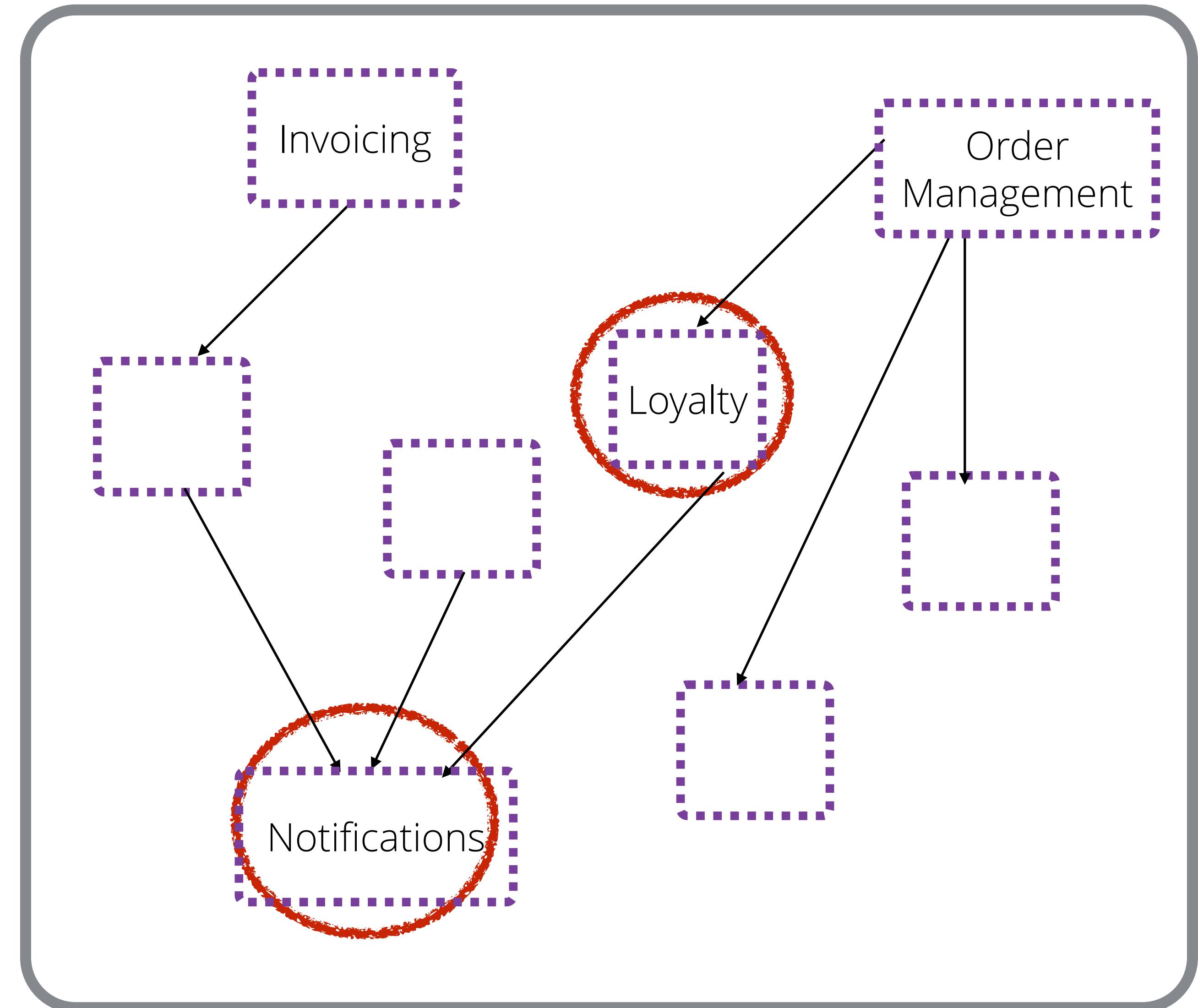
## WHAT ABOUT OTHER FUNCTIONALITY?

Can use “branch by abstraction” to help extraction of functionality deeper in the monolith

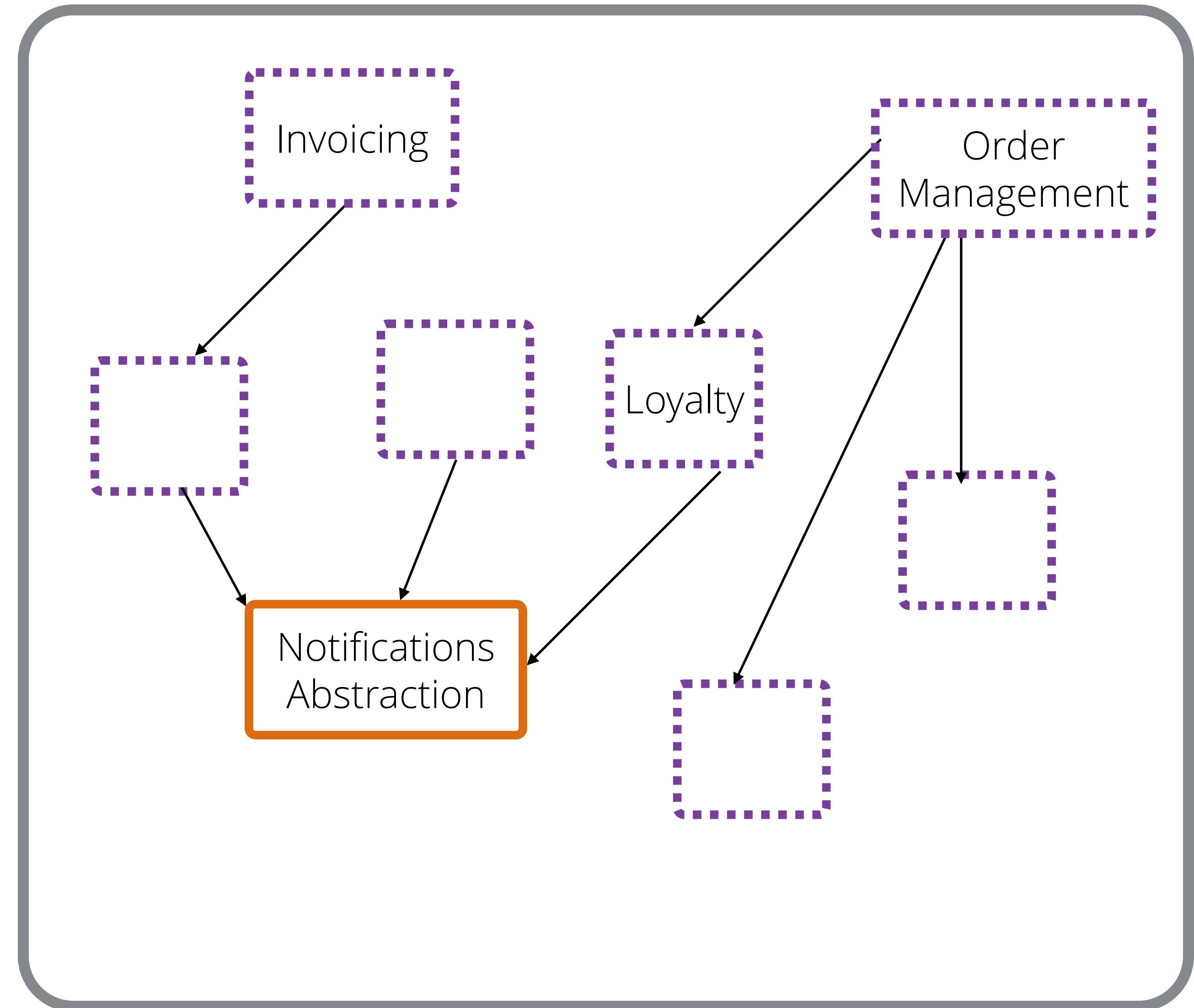


## WHAT ABOUT OTHER FUNCTIONALITY?

Can use “branch by abstraction” to help extraction of functionality deeper in the monolith

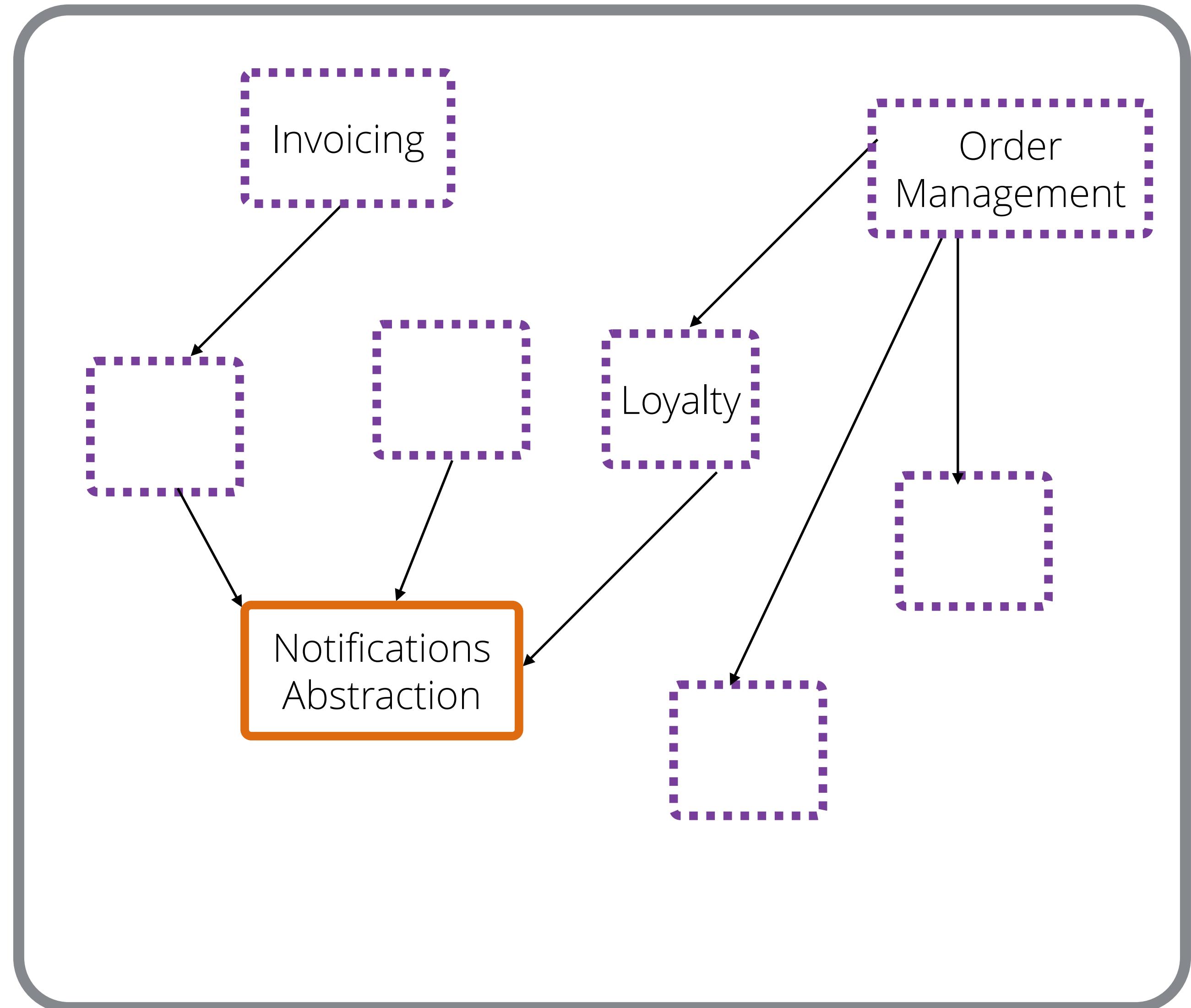


# BRANCH BY ABSTRACTION



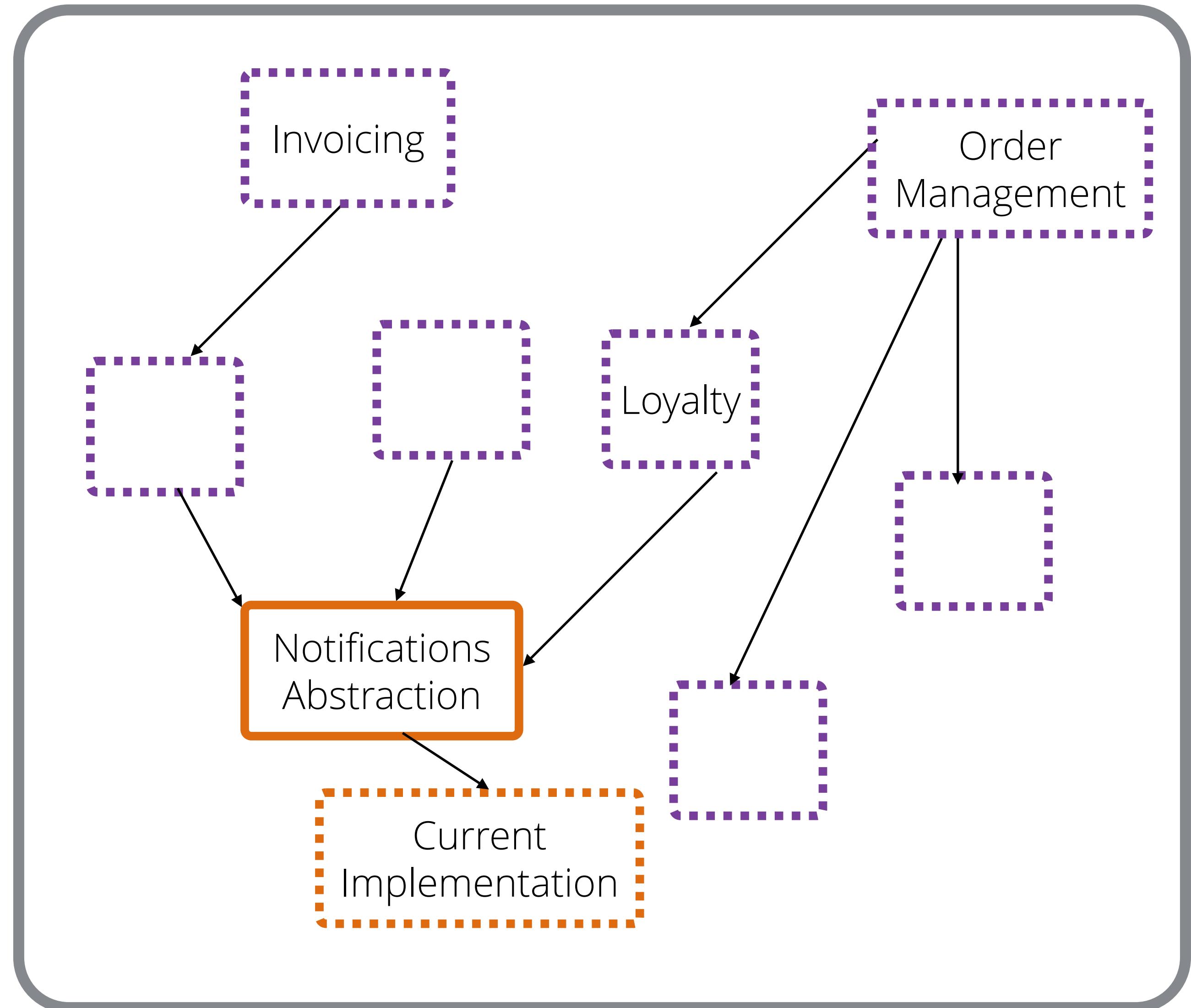
## BRANCH BY ABSTRACTION

Gives you an abstraction point which can be used to “hide” ongoing development



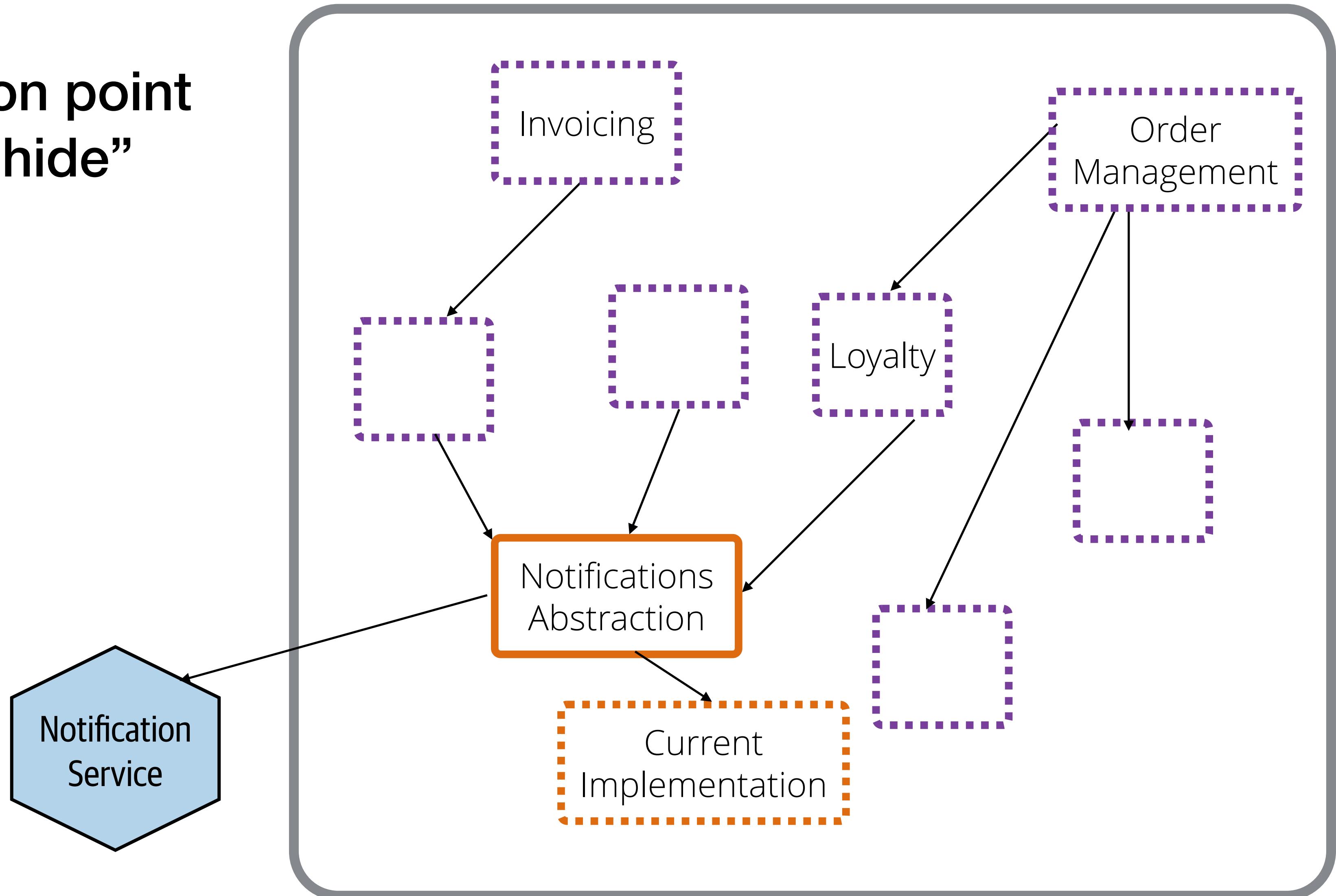
## BRANCH BY ABSTRACTION

Gives you an abstraction point which can be used to “hide” ongoing development



## BRANCH BY ABSTRACTION

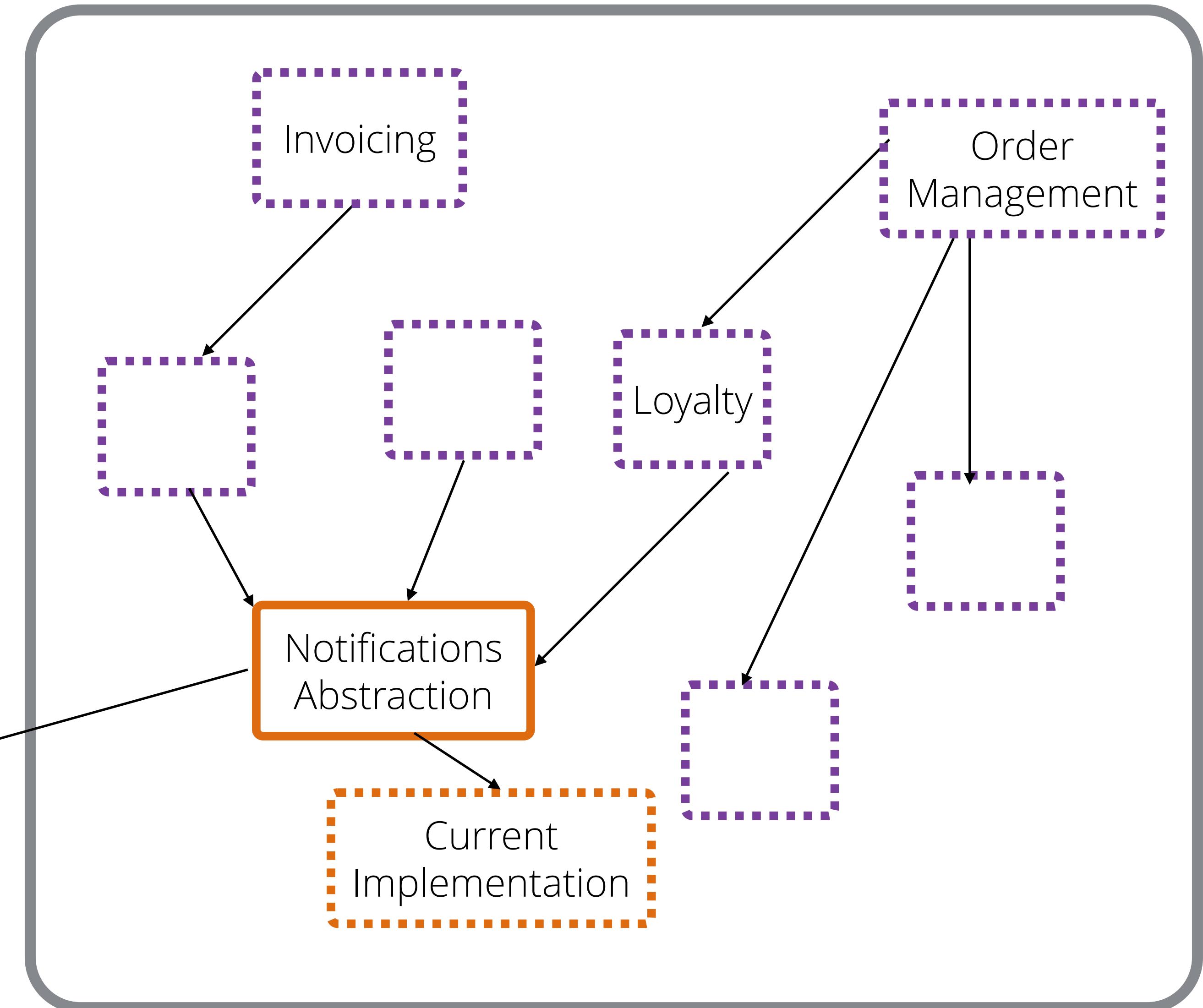
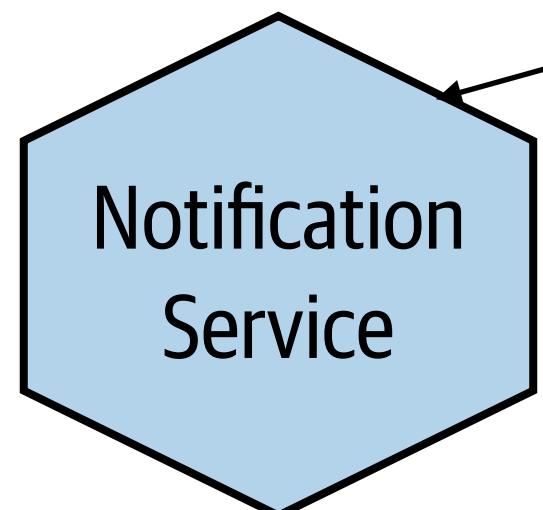
Gives you an abstraction point which can be used to “hide” ongoing development



## BRANCH BY ABSTRACTION

Gives you an abstraction point which can be used to “hide” ongoing development

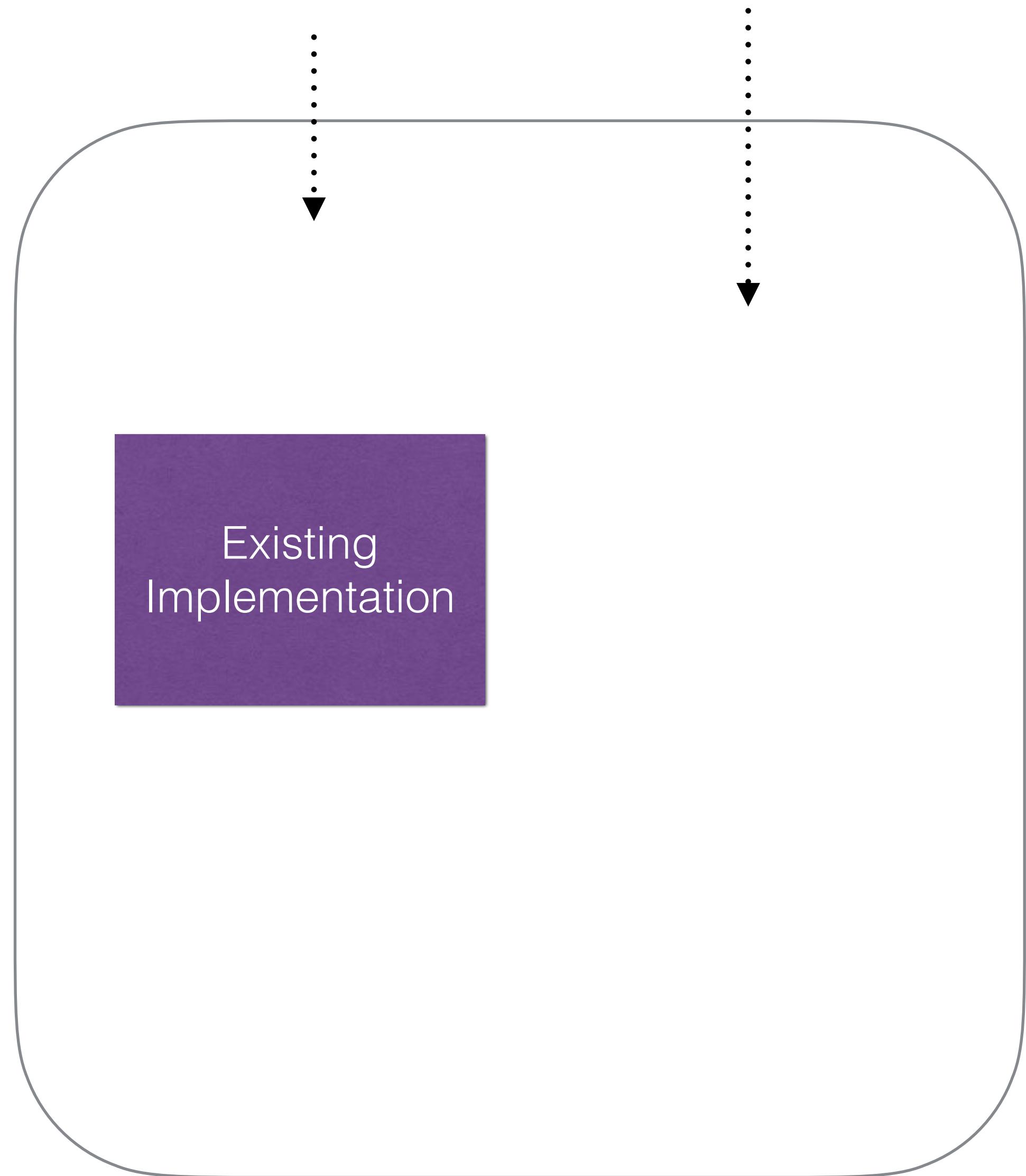
Can also be used as a way to toggle between implementations



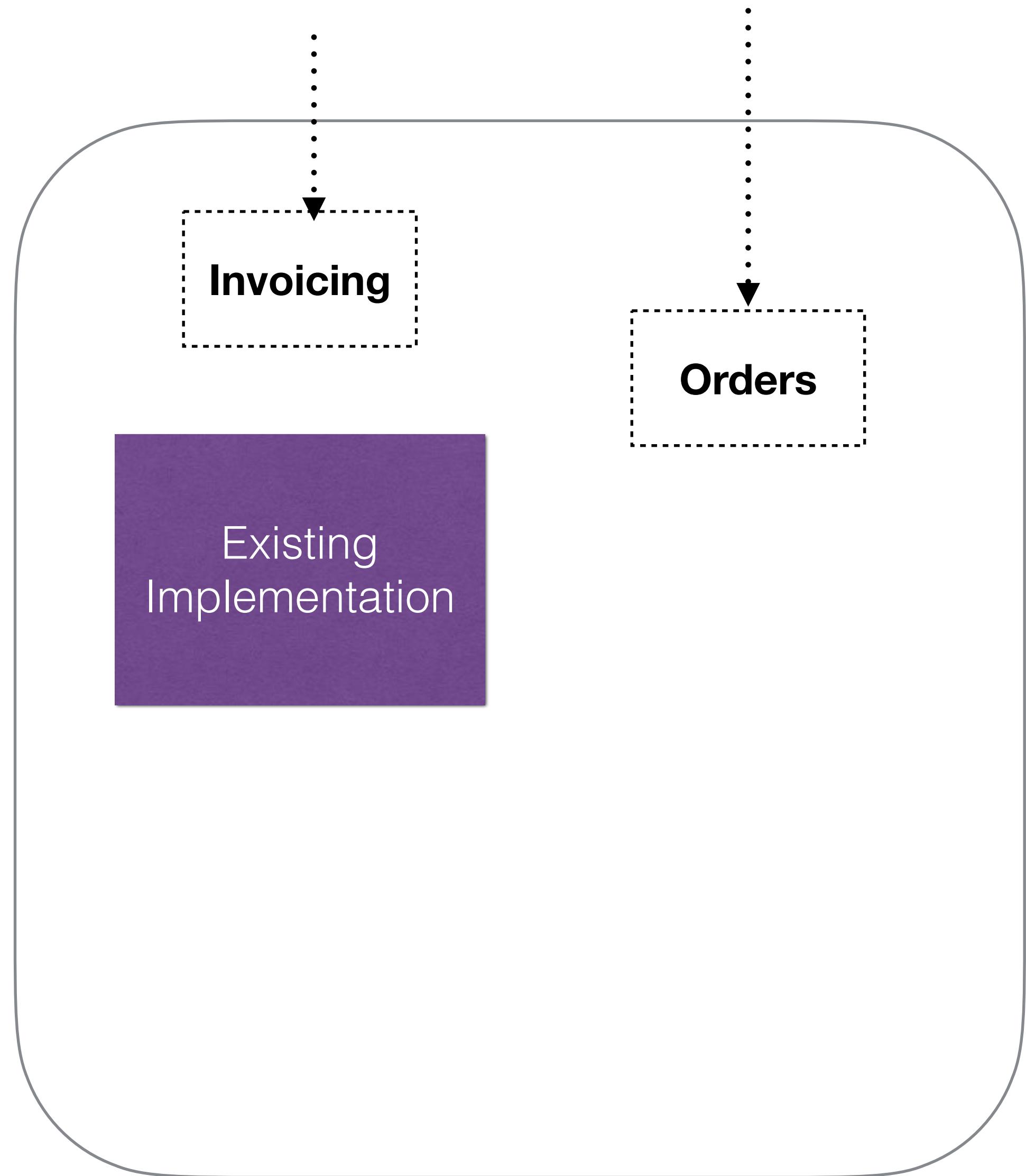
## EXAMPLE



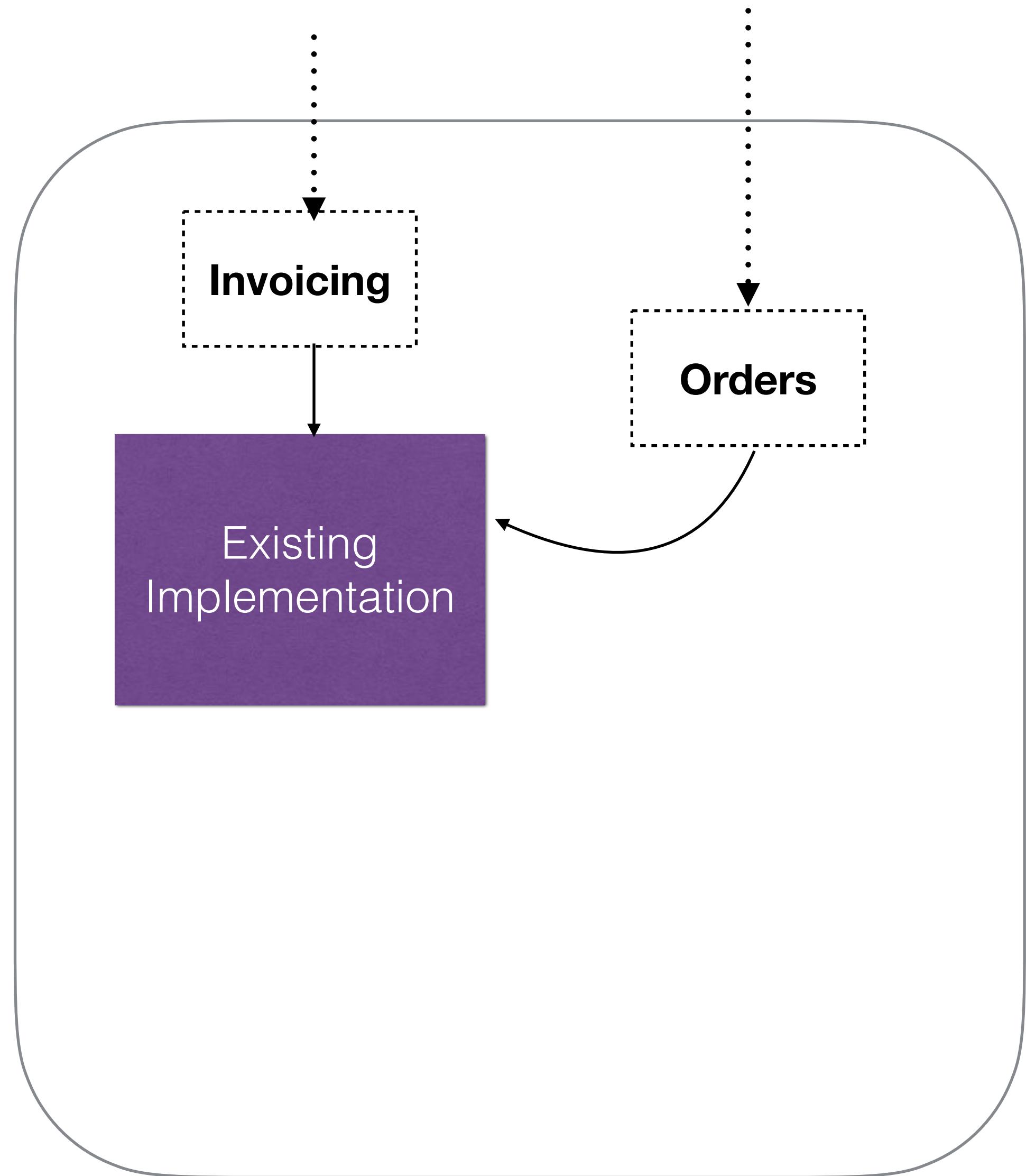
## EXAMPLE



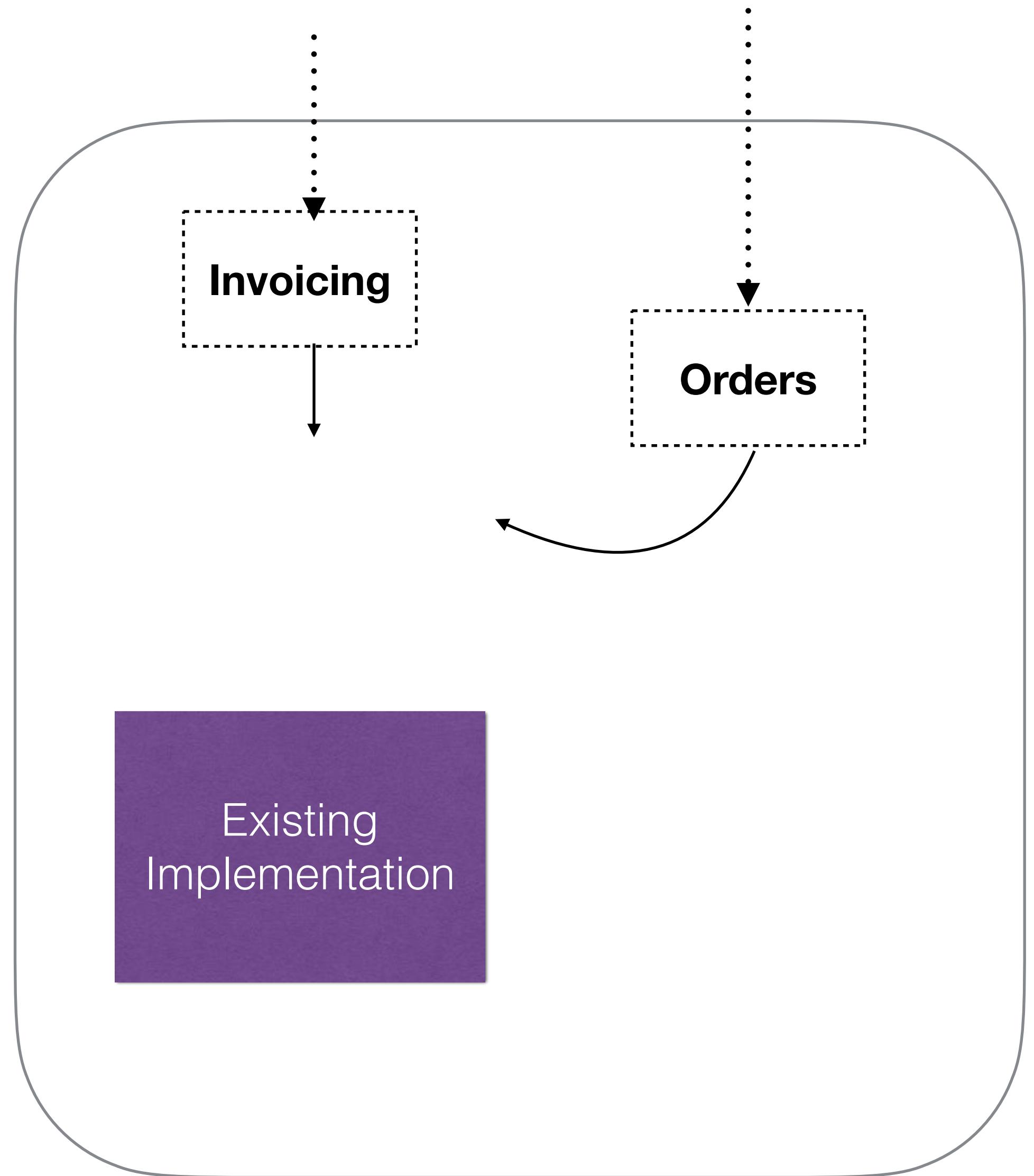
## EXAMPLE



## EXAMPLE

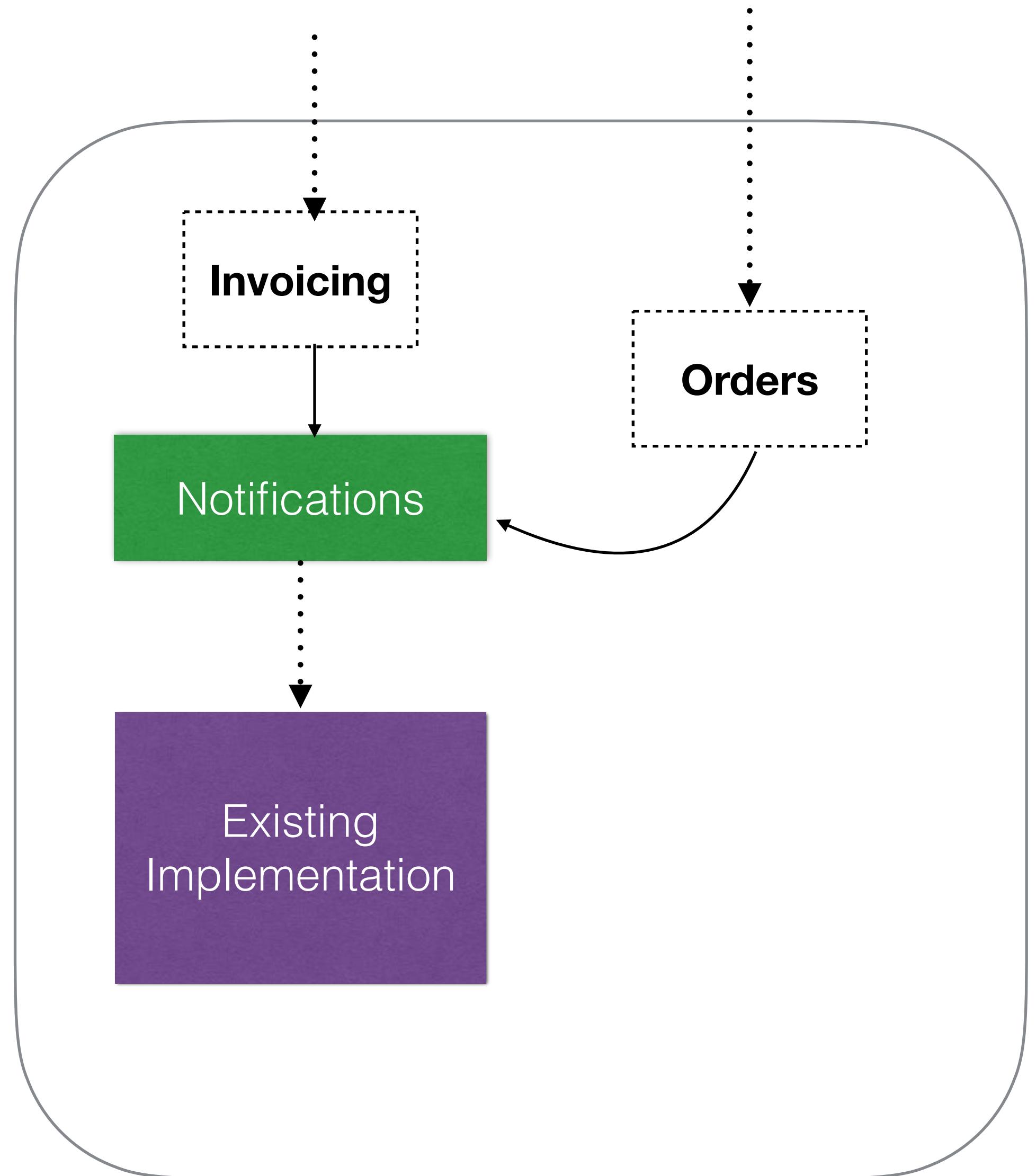


## EXAMPLE

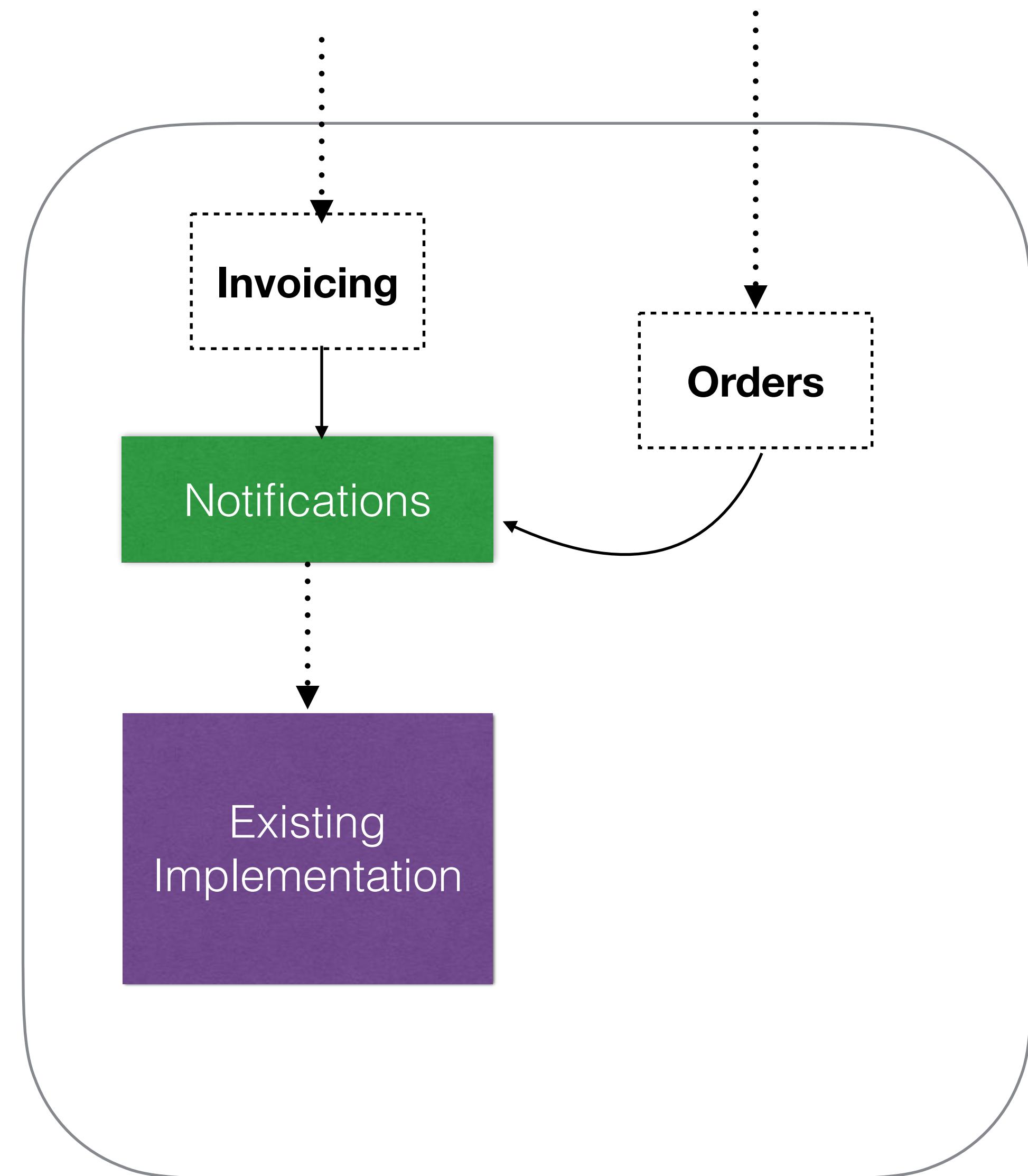


## EXAMPLE

### 1. Create abstraction point



## EXAMPLE

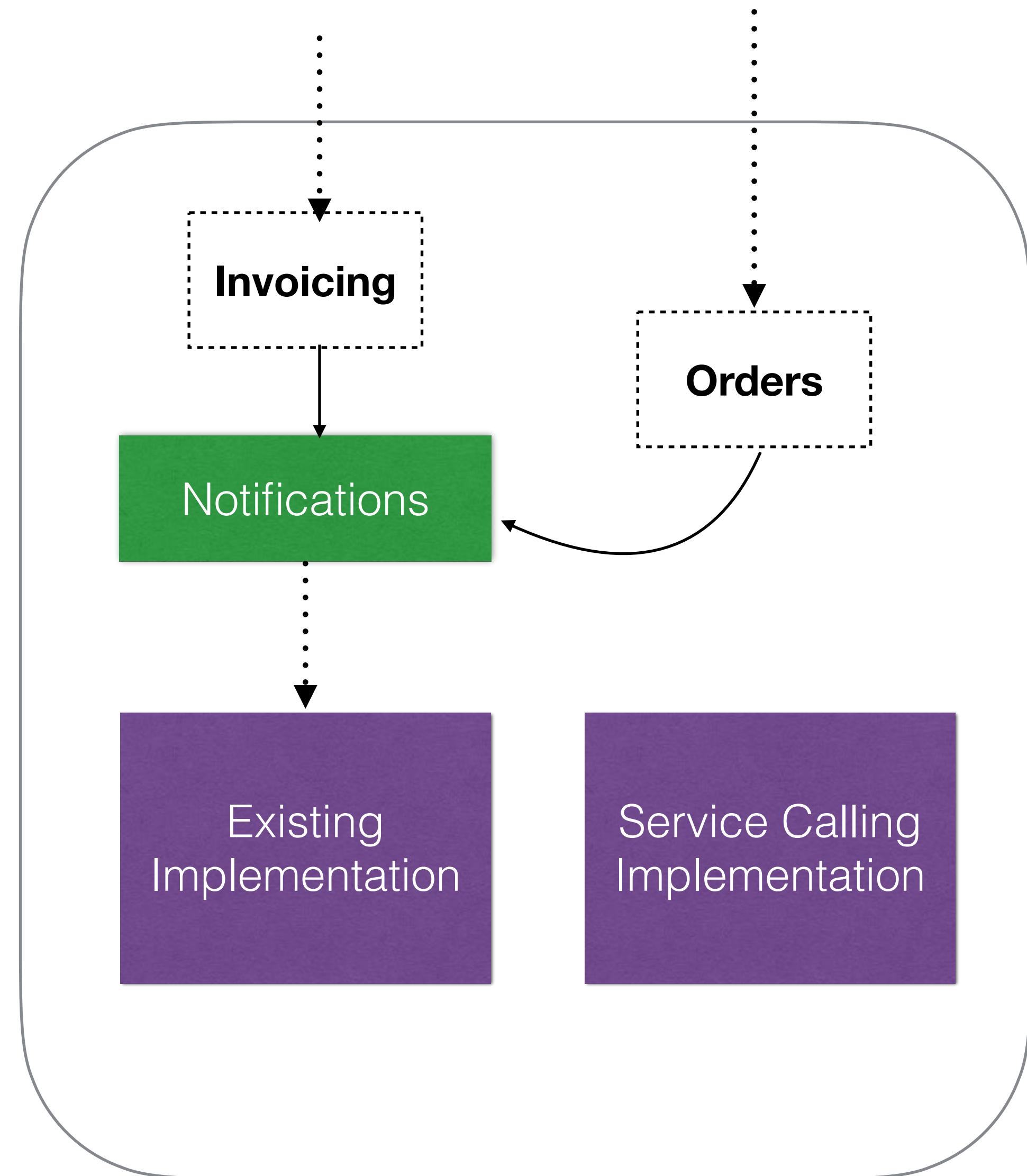


1. Create abstraction point
2. Start work on new service implementation

## EXAMPLE

1. Create abstraction point

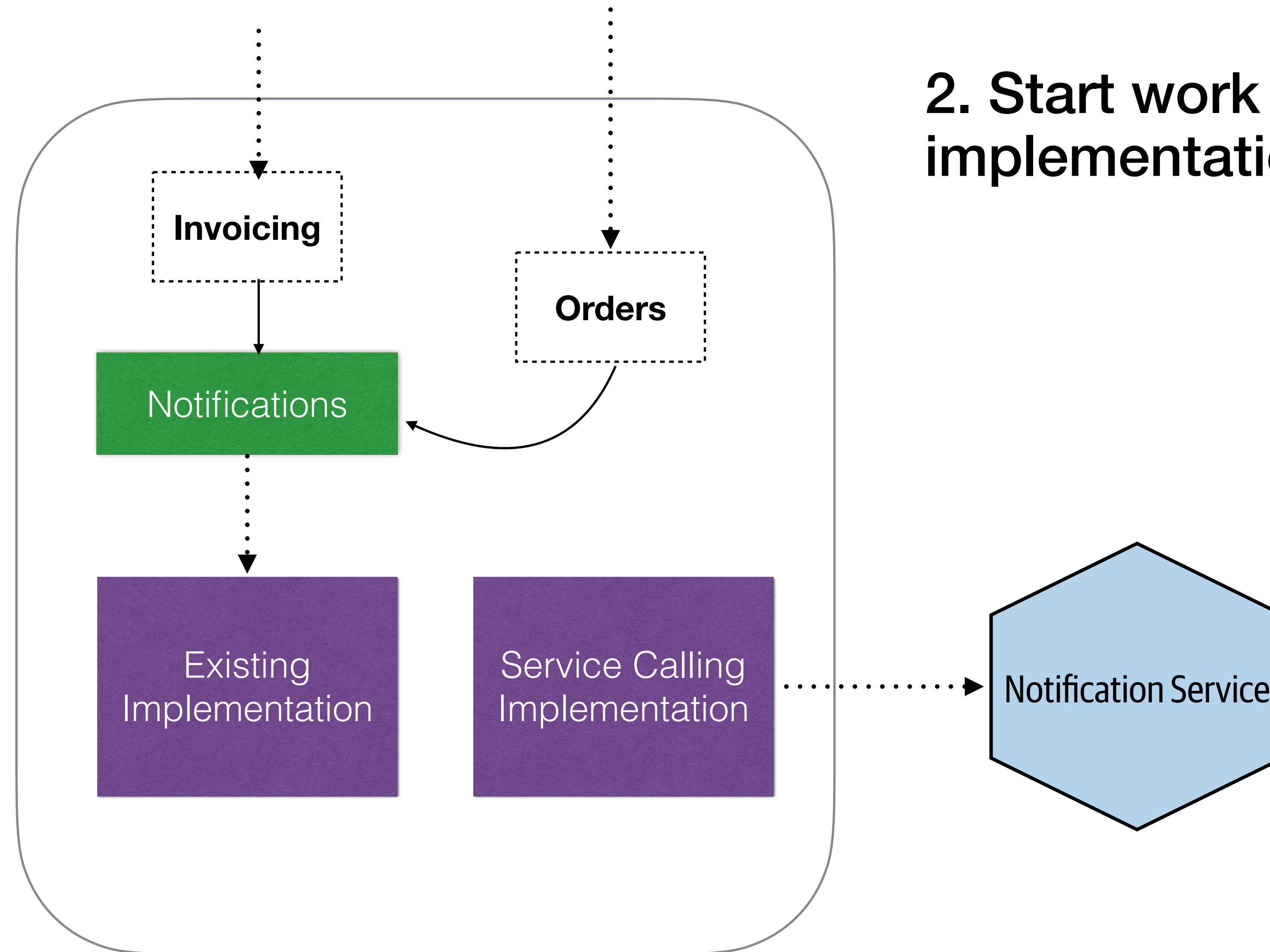
2. Start work on new service implementation



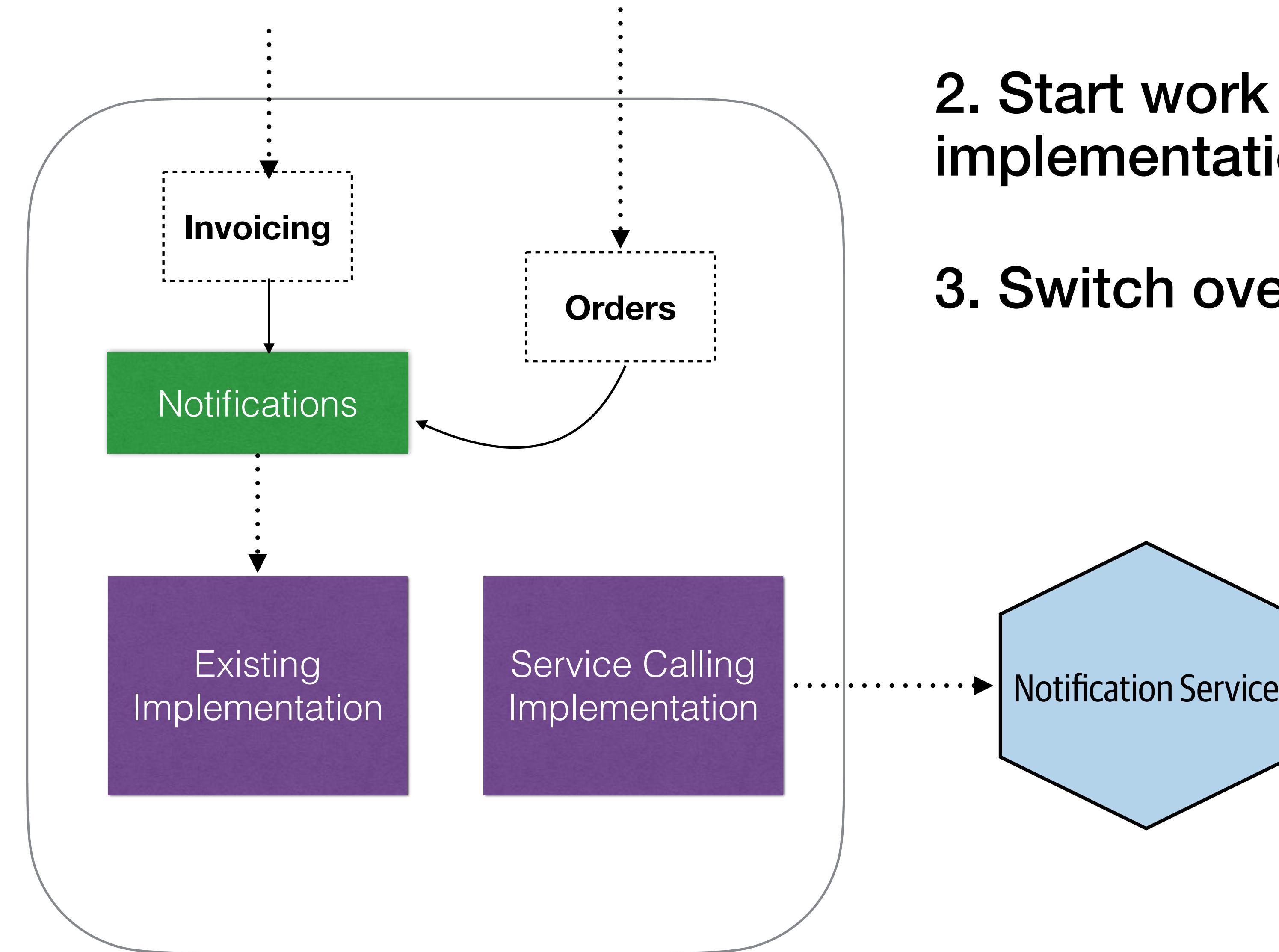
## EXAMPLE

1. Create abstraction point

2. Start work on new service implementation

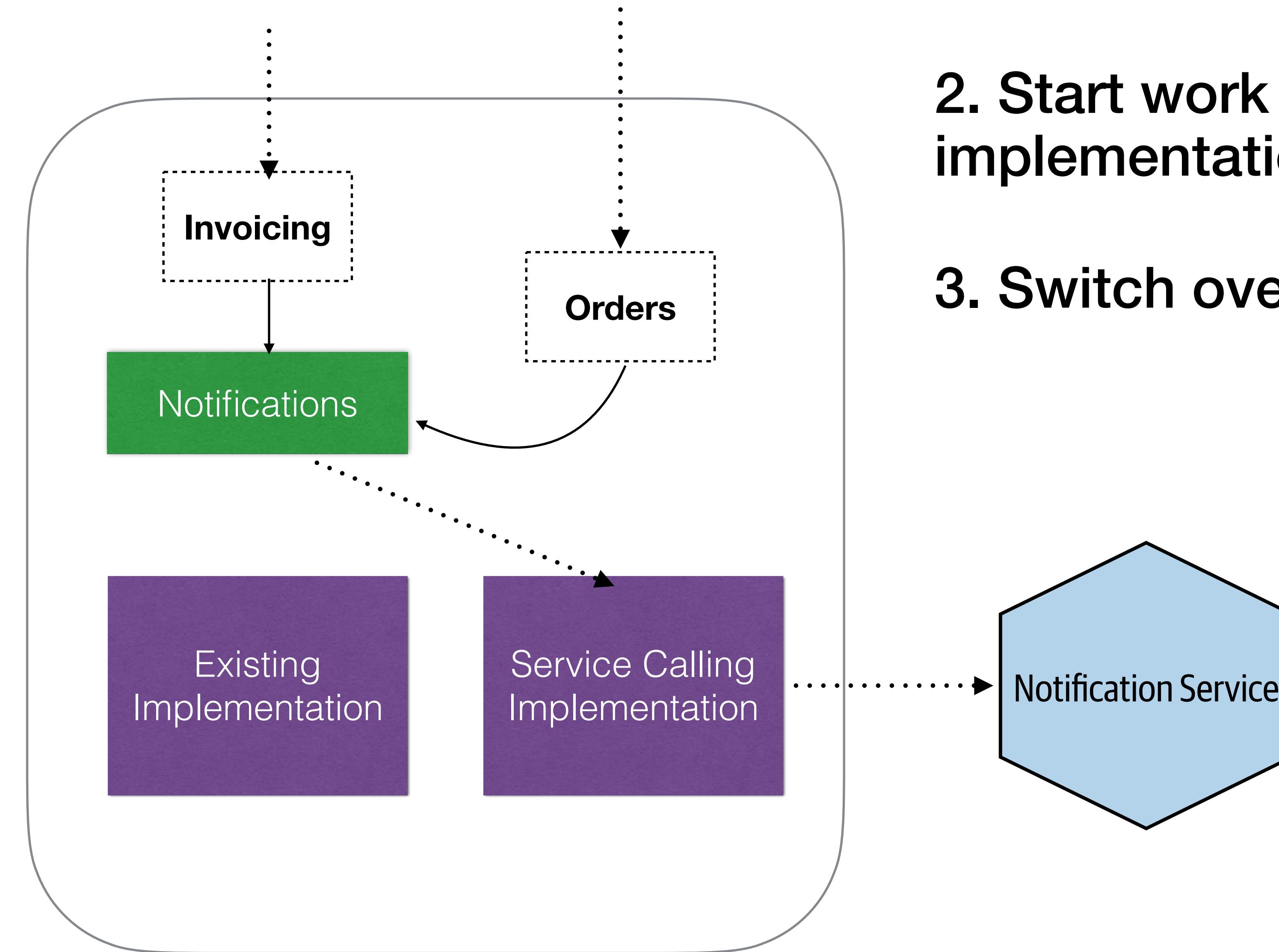


## EXAMPLE



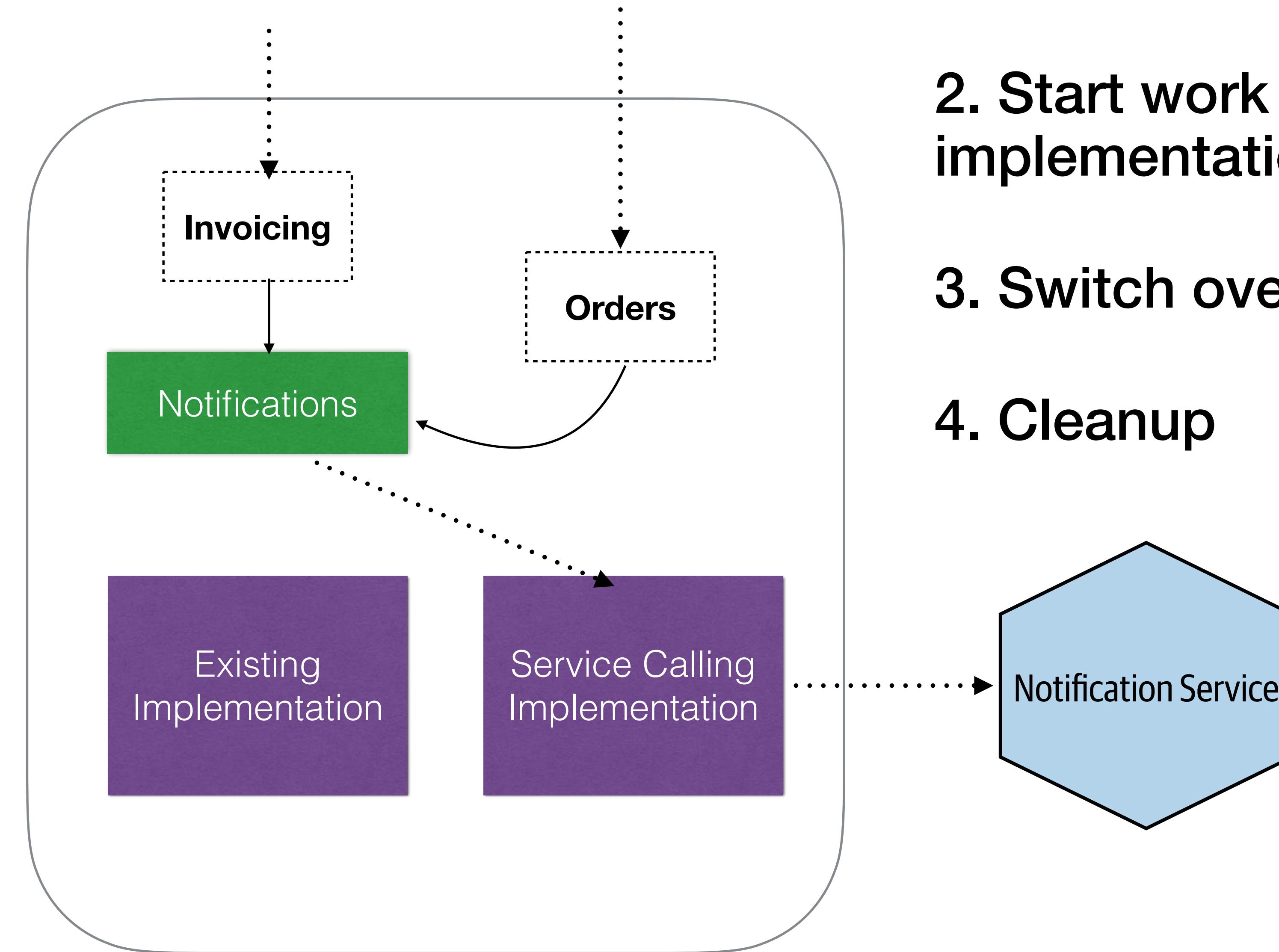
1. Create abstraction point
2. Start work on new service implementation
3. Switch over

## EXAMPLE



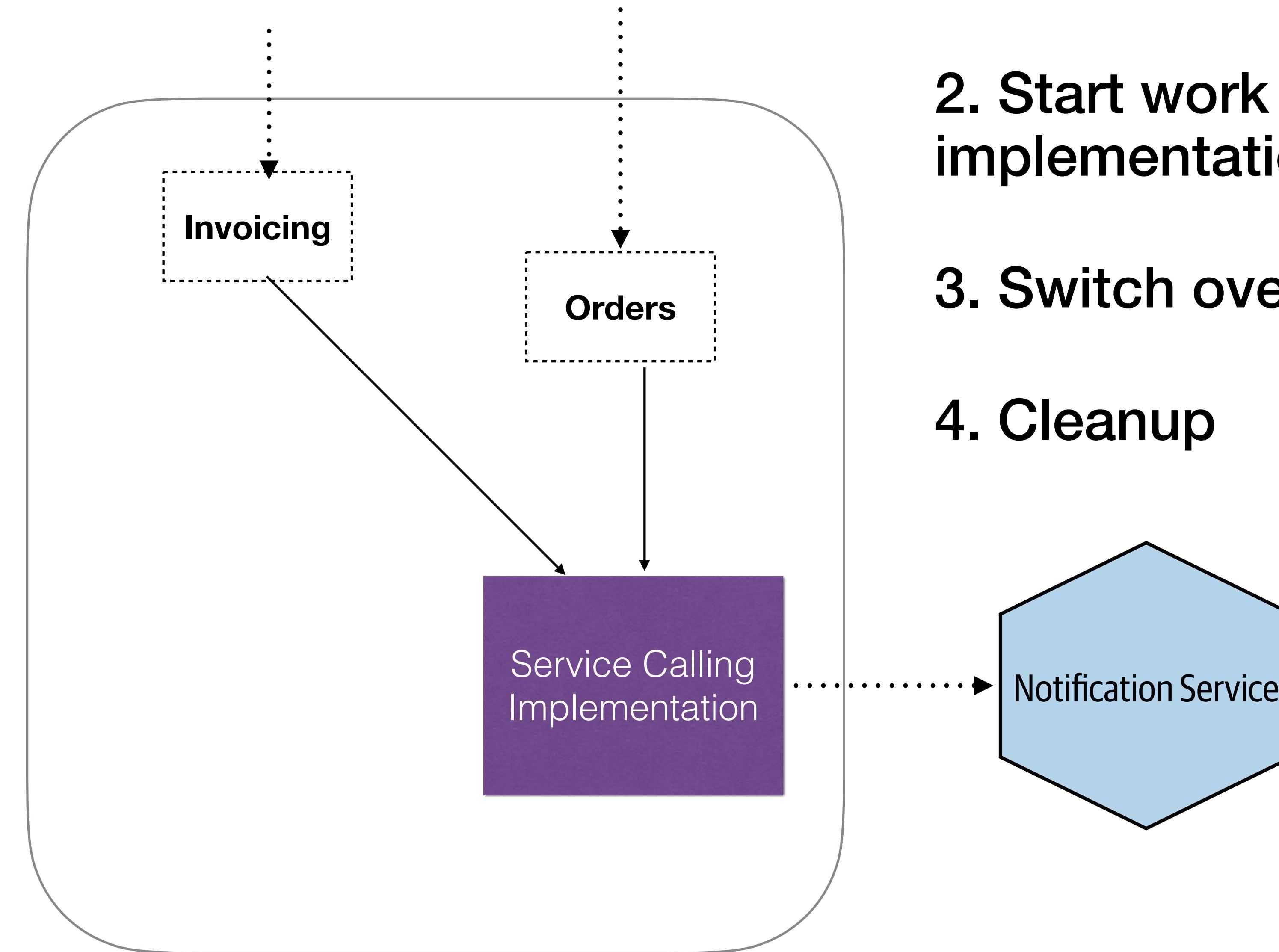
1. Create abstraction point
2. Start work on new service implementation
3. Switch over

## EXAMPLE

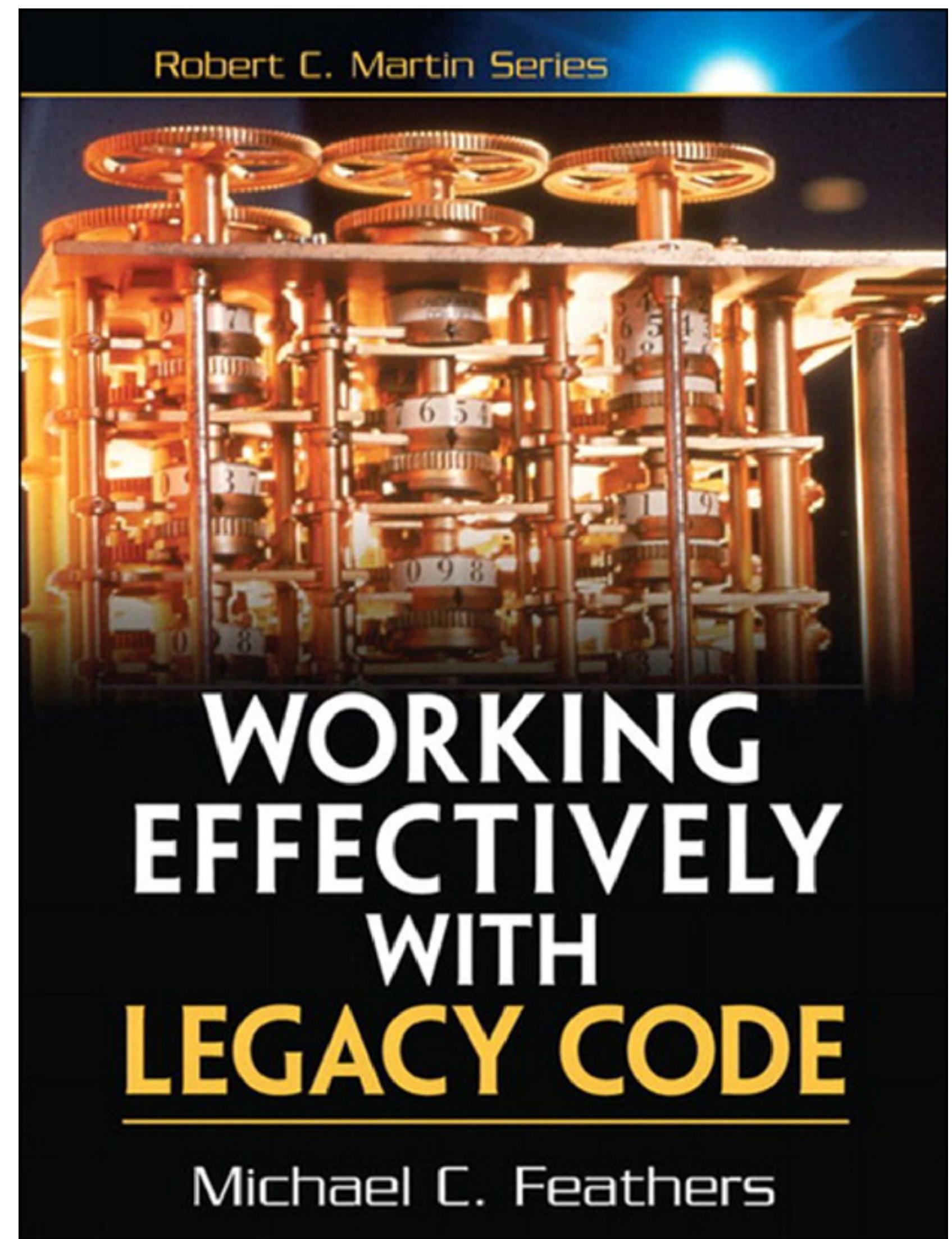


- 1. Create abstraction point**
- 2. Start work on new service implementation**
- 3. Switch over**
- 4. Cleanup**

## EXAMPLE



1. Create abstraction point
2. Start work on new service implementation
3. Switch over
4. Cleanup



## PATTERN: FEATURE TOGGLES



# PATTERN: FEATURE TOGGLES

## Feature Toggles (aka Feature Flags)

Feature Toggles (often also referred to as Feature Flags) are a powerful technique, allowing teams to modify system behavior without changing code. They fall into various usage categories, and it's important to take that categorization into account when implementing and managing toggles. Toggles introduce complexity. We can keep that complexity in check by using smart toggle implementation practices and appropriate tools to manage our toggle configuration, but we should also aim to constrain the number of toggles in our system.

09 October 2017



### CONTENTS

[expand](#)

- [A Toggling Tale](#)
- [Categories of toggles](#)
- [Implementation Techniques](#)
- [Toggle Configuration](#)
- [Working with feature-flagged systems](#)

Pete Hodgson

Pete Hodgson is an independent  
software delivery consultant based in  
the San Francisco Bay Area. He  
specializes in helping startup

## Allow for functionality to be toggled off and on

# PATTERN: FEATURE TOGGLES

## Feature Toggles (aka Feature Flags)

Feature Toggles (often also referred to as Feature Flags) are a powerful technique, allowing teams to modify system behavior without changing code. They fall into various usage categories, and it's important to take that categorization into account when implementing and managing toggles. Toggles introduce complexity. We can keep that complexity in check by using smart toggle implementation practices and appropriate tools to manage our toggle configuration, but we should also aim to constrain the number of toggles in our system.

09 October 2017



### CONTENTS

[expand](#)

- [A Toggling Tale](#)
- [Categories of toggles](#)
- [Implementation Techniques](#)
- [Toggle Configuration](#)
- [Working with feature-flagged systems](#)

Pete Hodgson

Pete Hodgson is an independent  
software delivery consultant based in  
the San Francisco Bay Area. He  
specializes in helping startup

**Allow for functionality to be toggled off and on**

**Can also be used to switch between alternative implementations**

# PATTERN: FEATURE TOGGLES

## Feature Toggles (aka Feature Flags)

Feature Toggles (often also referred to as Feature Flags) are a powerful technique, allowing teams to modify system behavior without changing code. They fall into various usage categories, and it's important to take that categorization into account when implementing and managing toggles. Toggles introduce complexity. We can keep that complexity in check by using smart toggle implementation practices and appropriate tools to manage our toggle configuration, but we should also aim to constrain the number of toggles in our system.

09 October 2017



Pete Hodgson

Pete Hodgson is an independent software delivery consultant based in the San Francisco Bay Area. He specializes in helping startup

### CONTENTS

expand

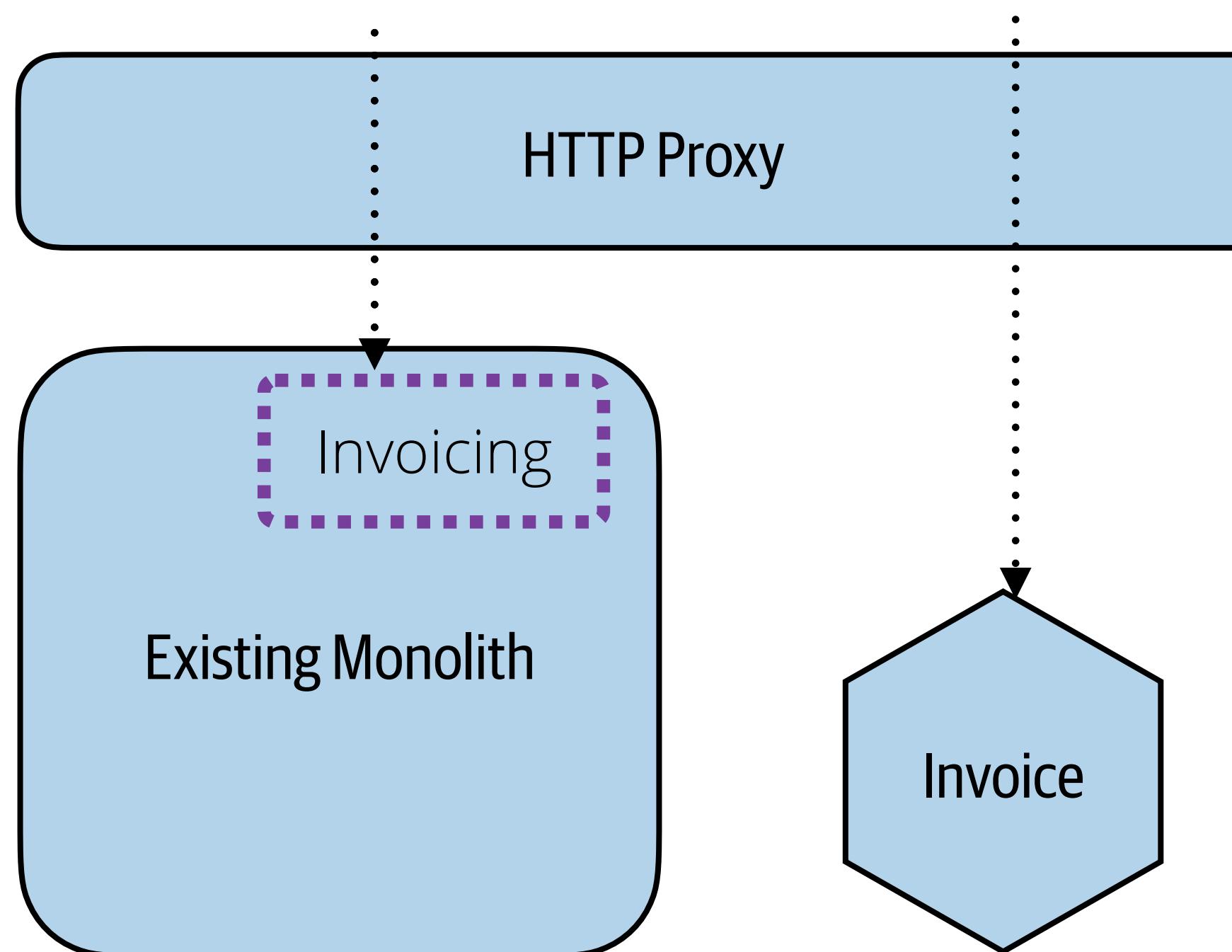
- [A Toggling Tale](#)
- [Categories of toggles](#)
- [Implementation Techniques](#)
- [Toggle Configuration](#)
- [Working with feature-flagged systems](#)

**Allow for functionality to be toggled off and on**

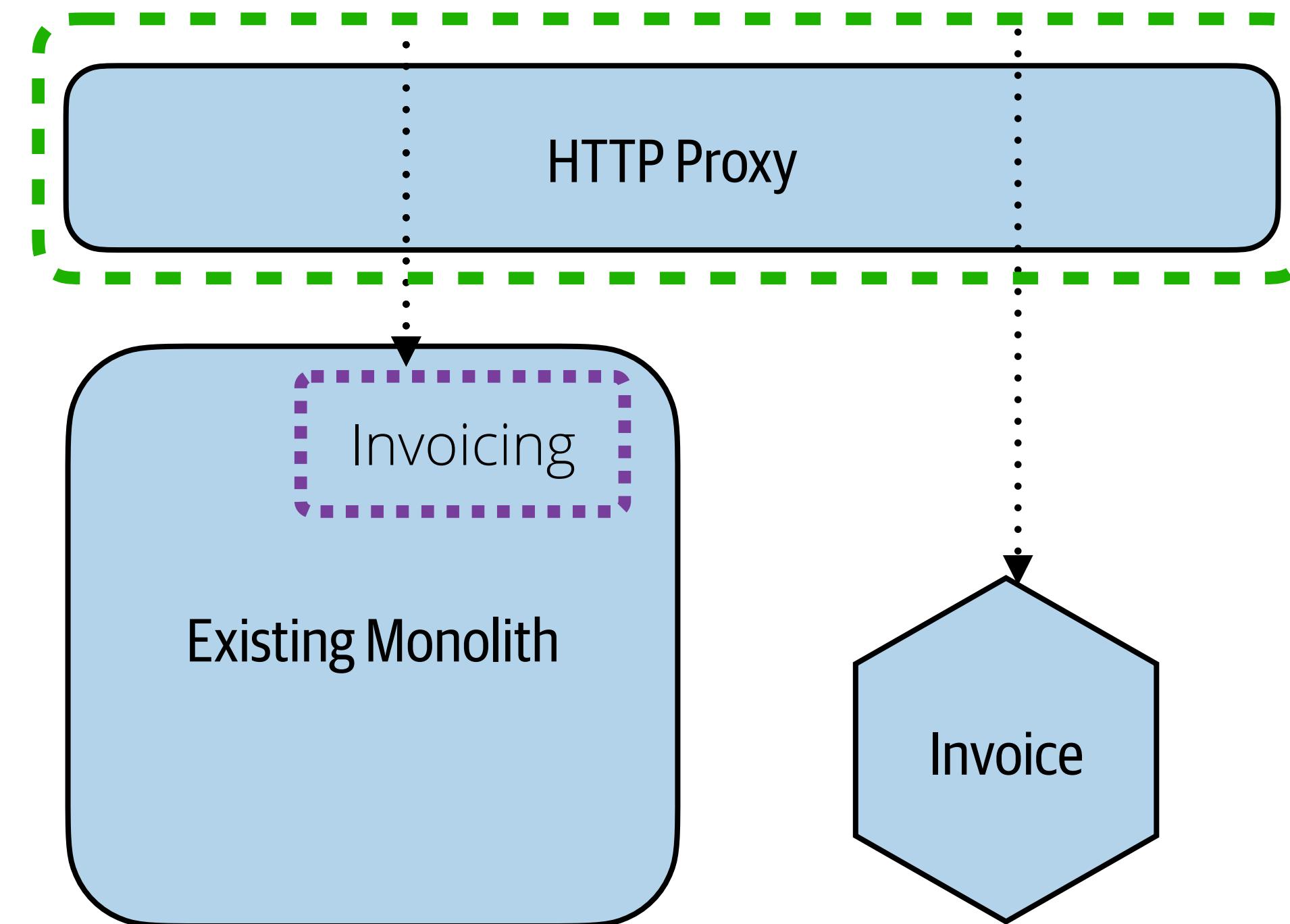
**Can also be used to switch between alternative implementations**

**Changing toggled values doesn't require a rebuild or redeploy**

## WITH THE STRANGLER FIG PATTERN

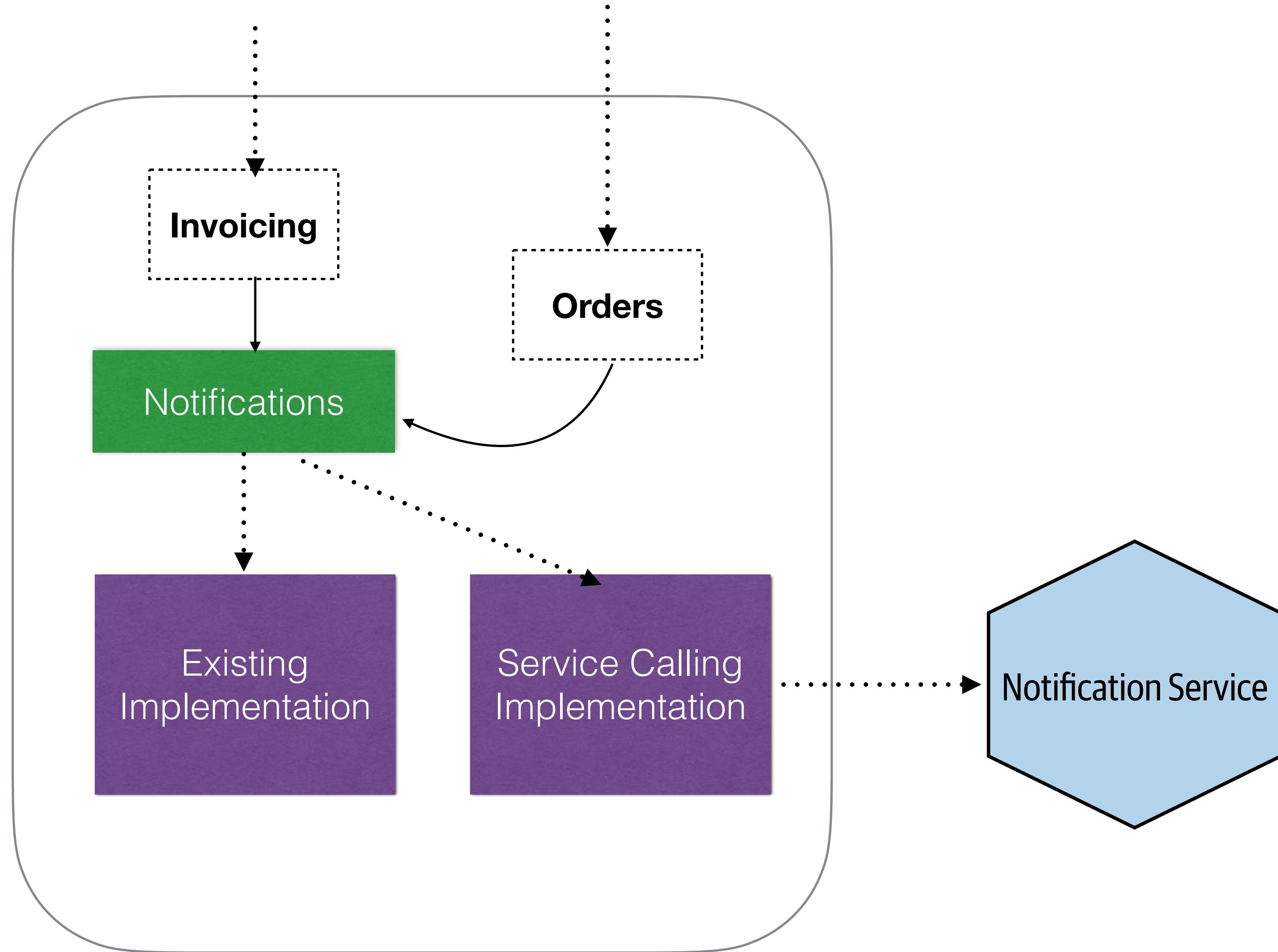


## WITH THE STRANGLER FIG PATTERN



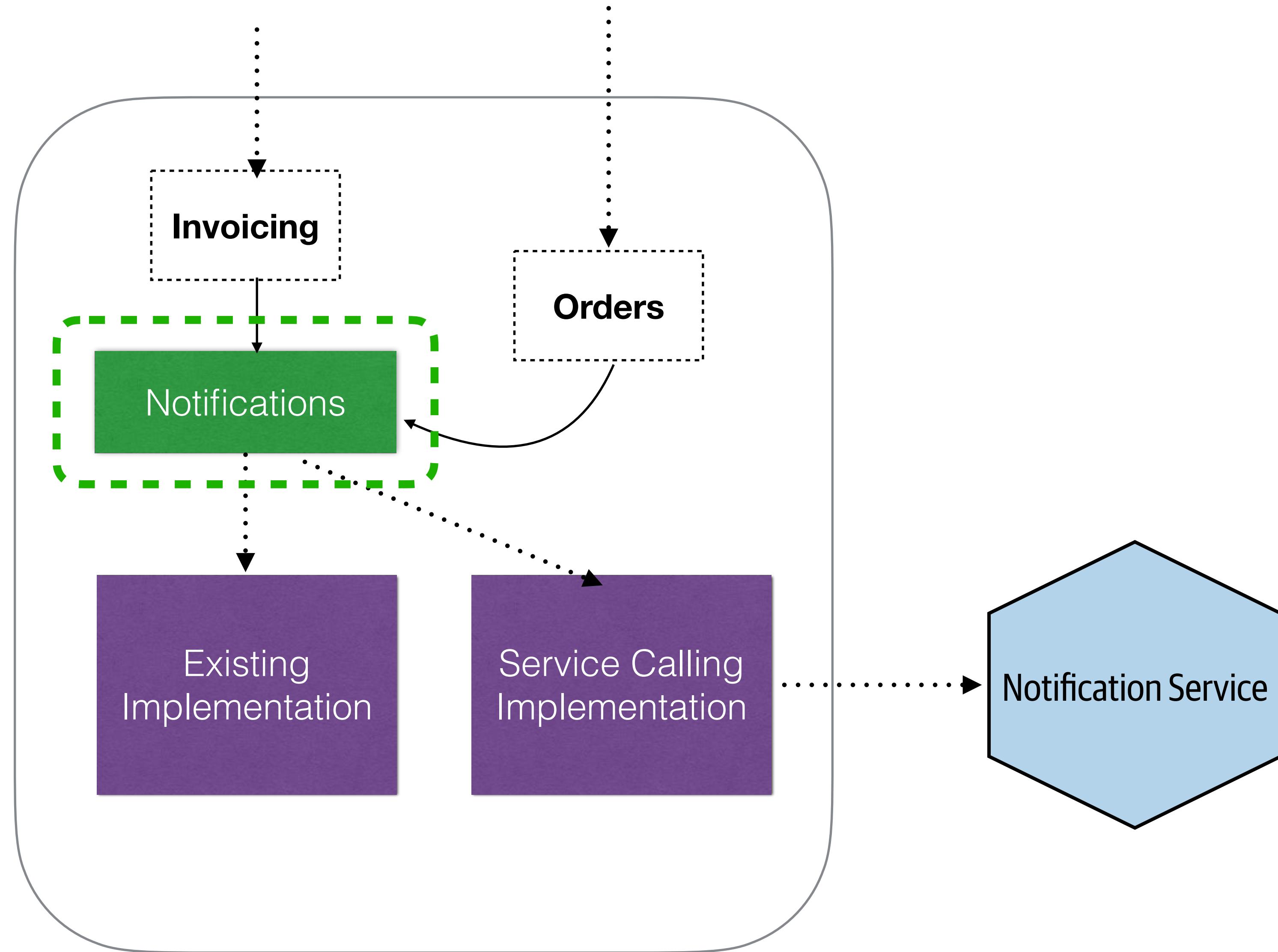
**Implement a simple config-based toggle in the HTTP Proxy configuration**

## WITH BRANCH BY ABSTRACTION



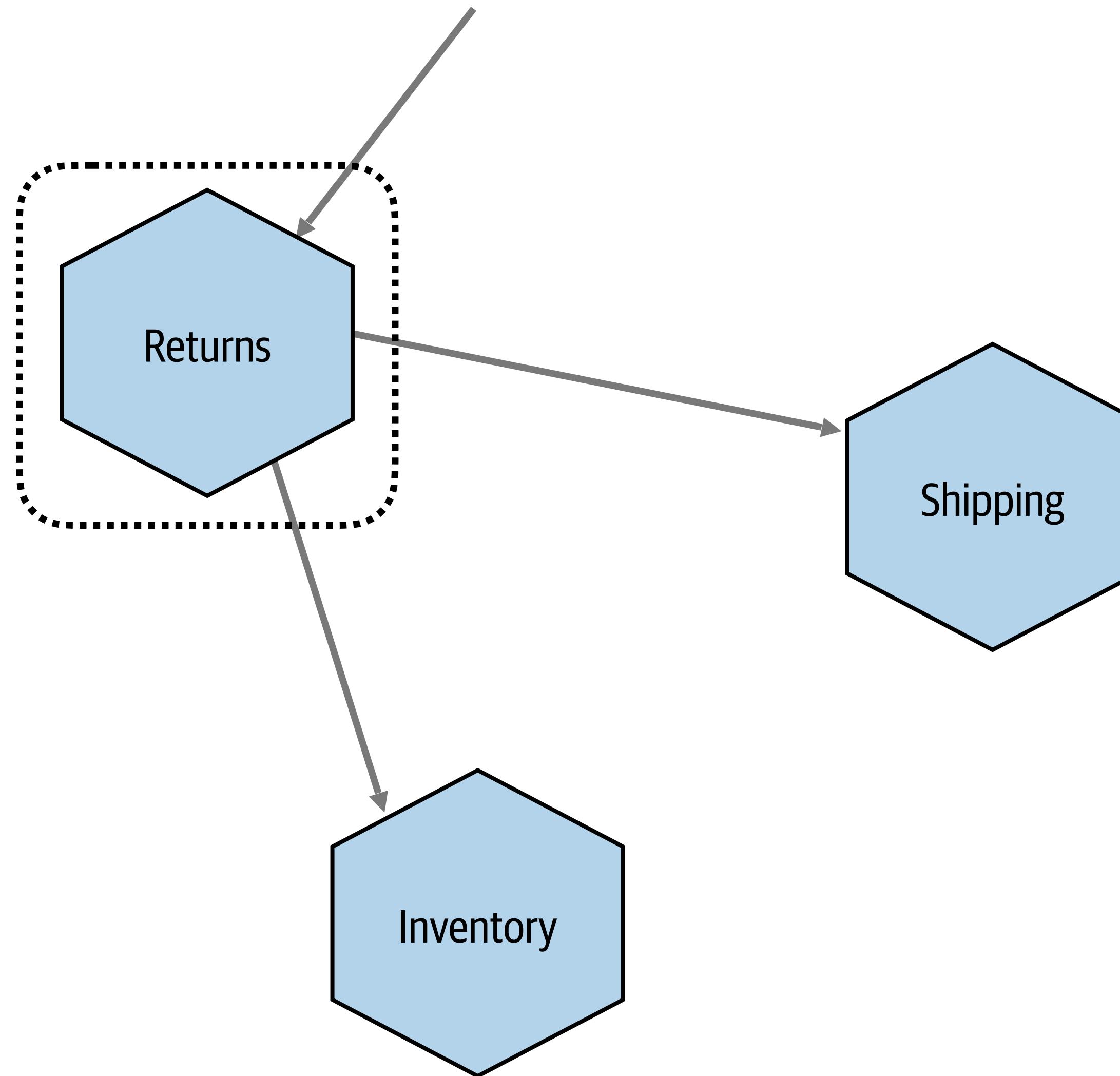
## WITH BRANCH BY ABSTRACTION

Use the abstraction to divert calls based on toggle value

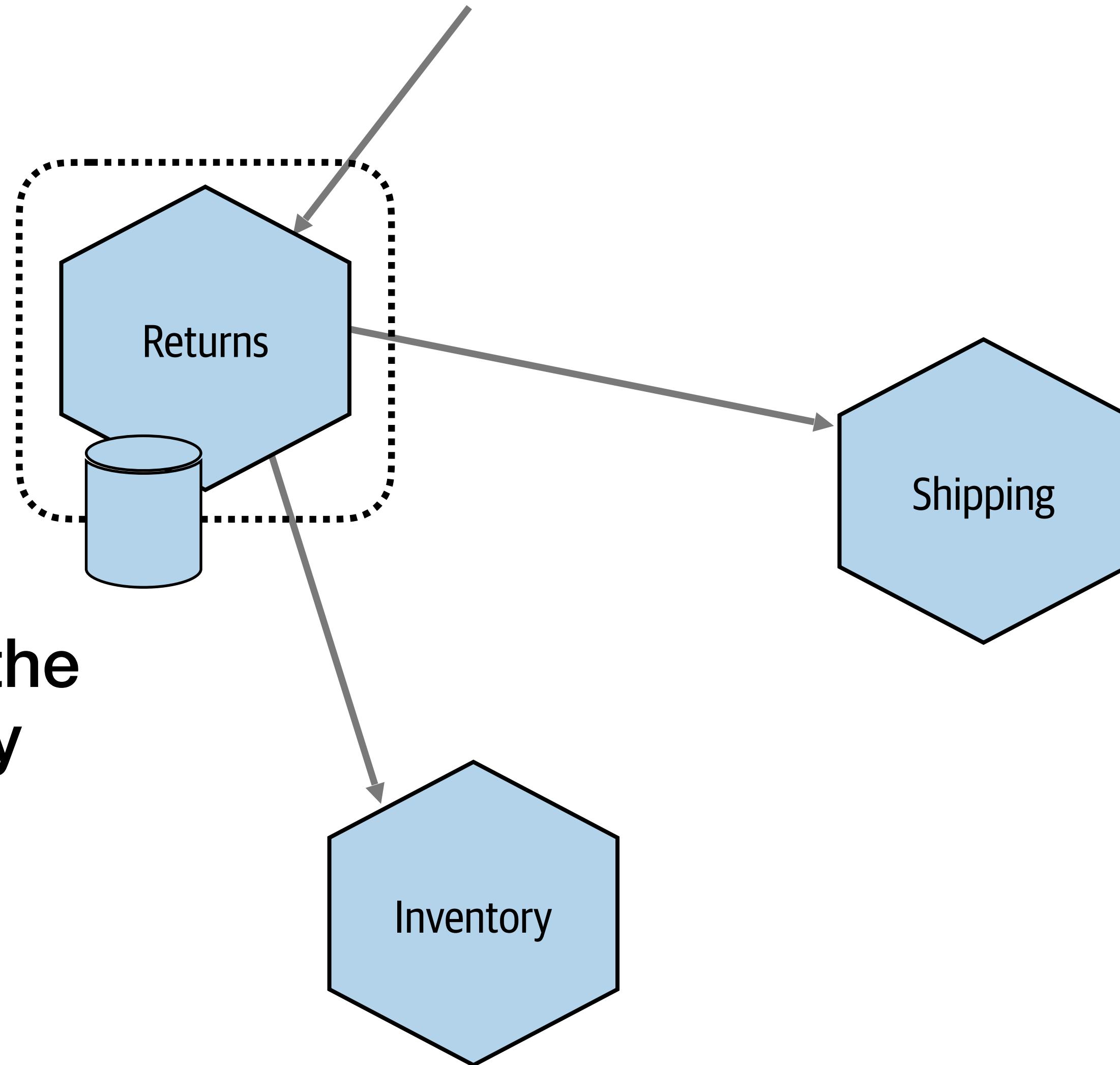


# **3/4 Hiding Data**

## DATA IS HIDDEN...

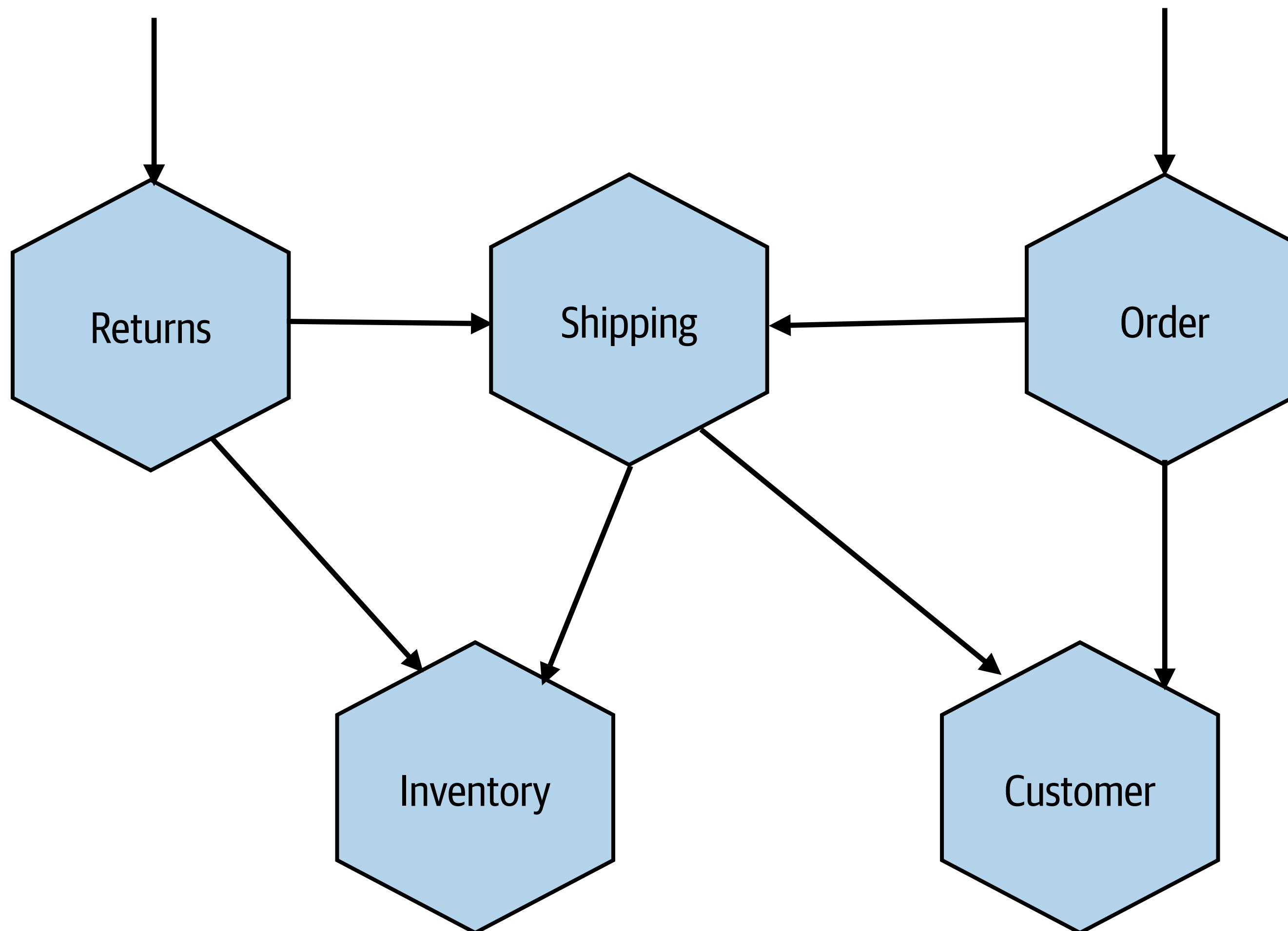


## DATA IS HIDDEN...



**Data is hidden inside the microservice boundary**

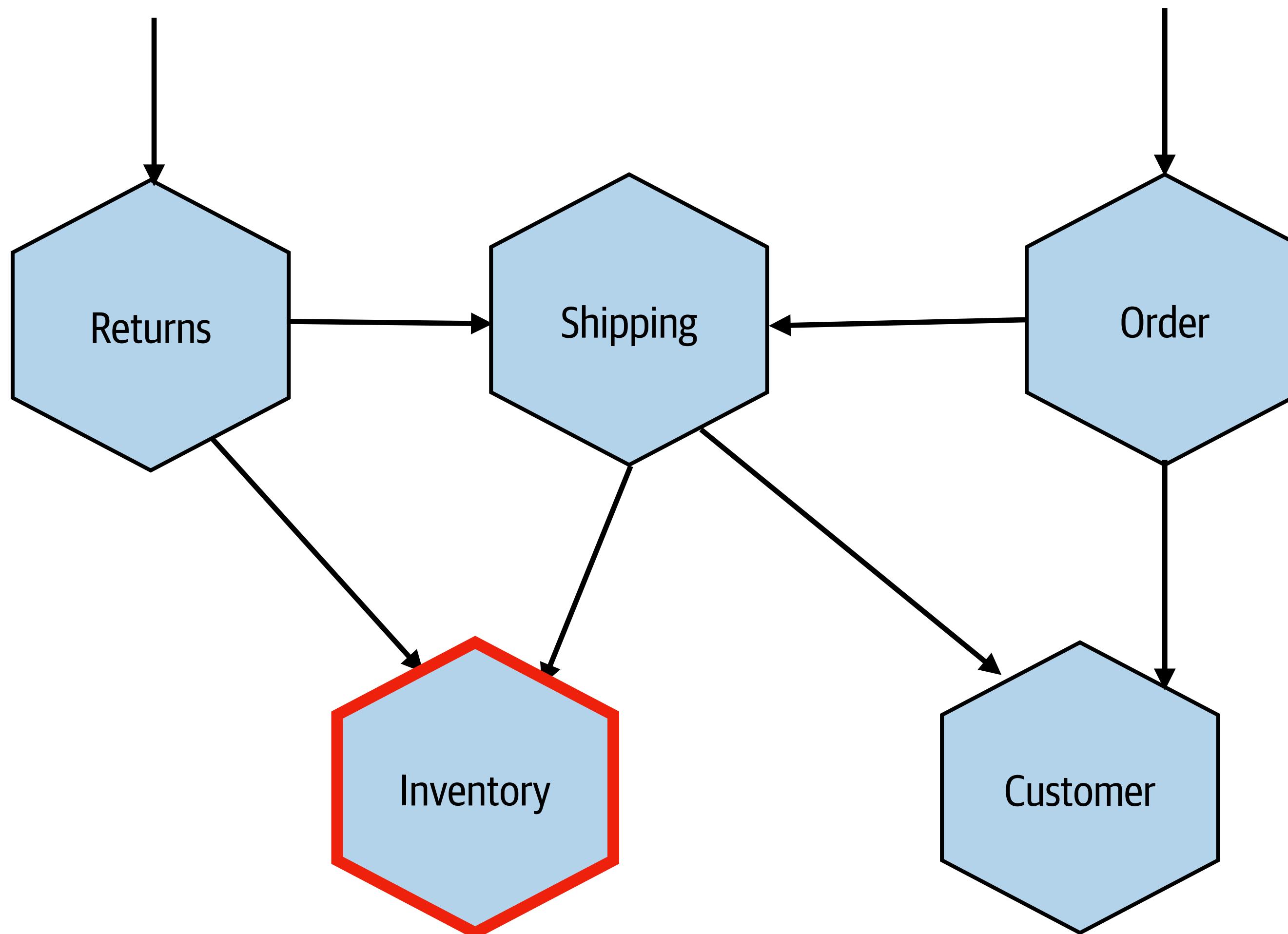
## INDEPENDENT DEPLOYABILITY



**When we make a change, we need to make sure we don't break upstream consumers**

**So backwards compatibility is key**

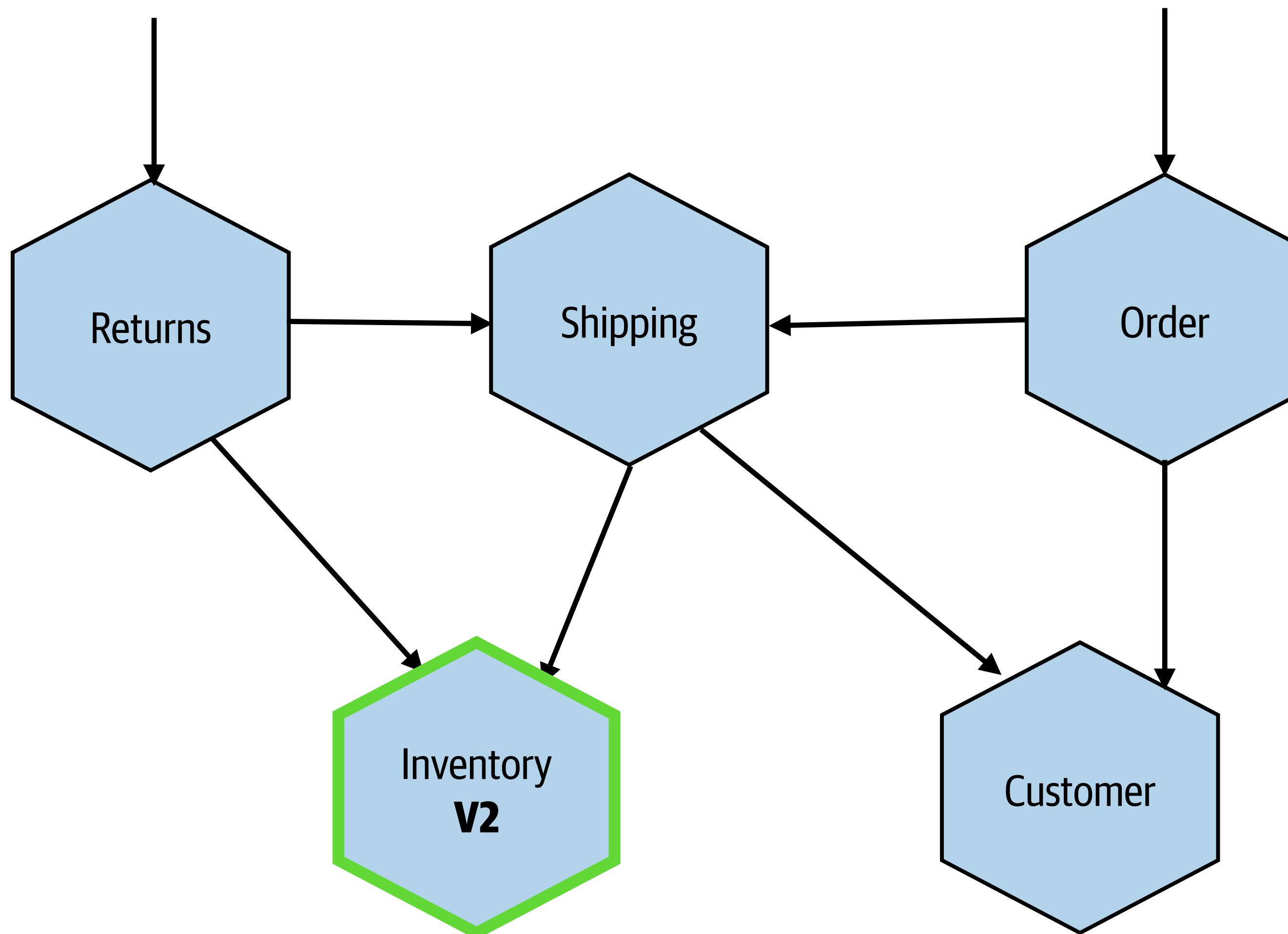
## INDEPENDENT DEPLOYABILITY



**When we make a change, we need to make sure we don't break upstream consumers**

**So backwards compatibility is key**

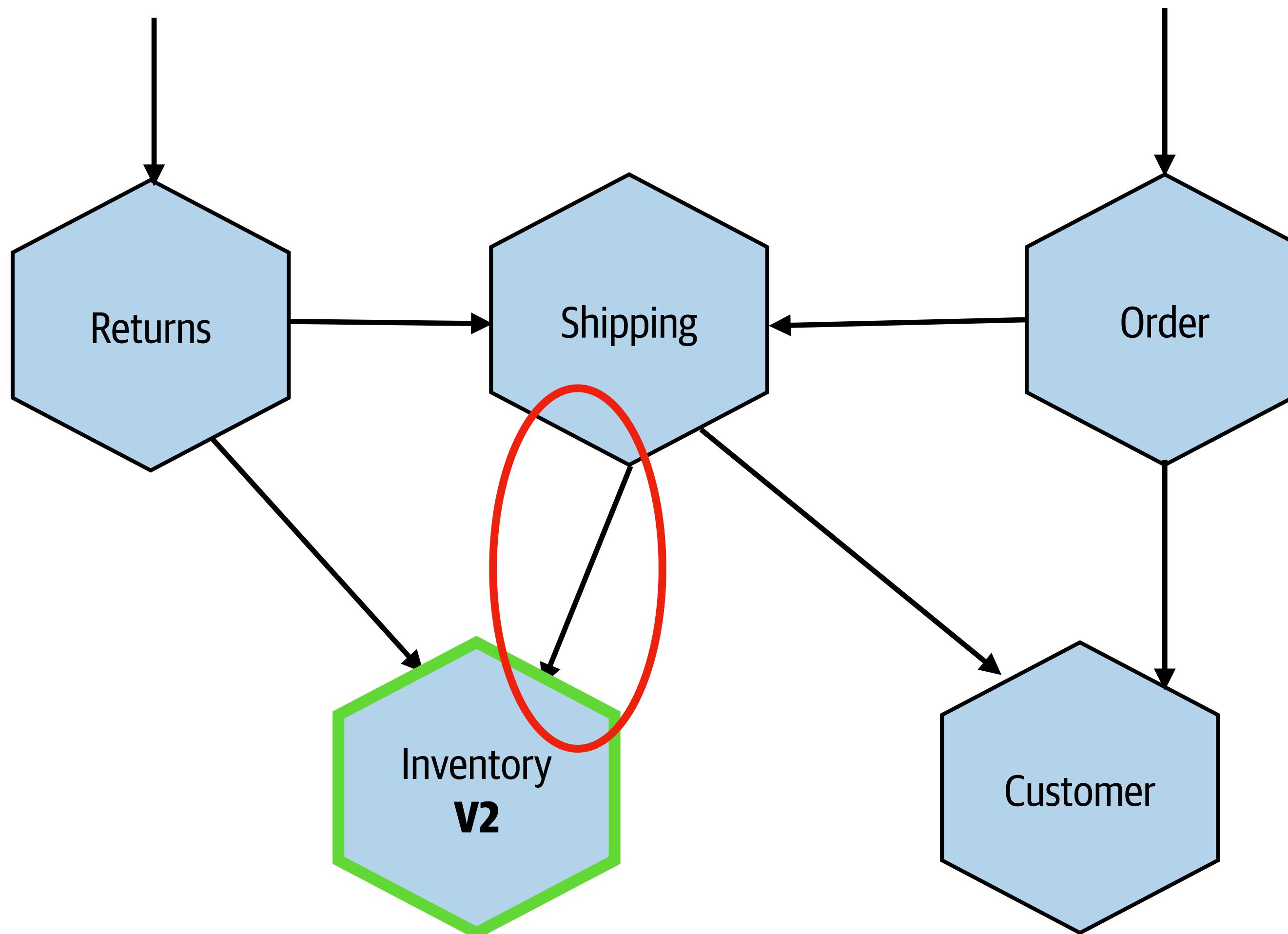
## INDEPENDENT DEPLOYABILITY



**When we make a change, we need to make sure we don't break upstream consumers**

**So backwards compatibility is key**

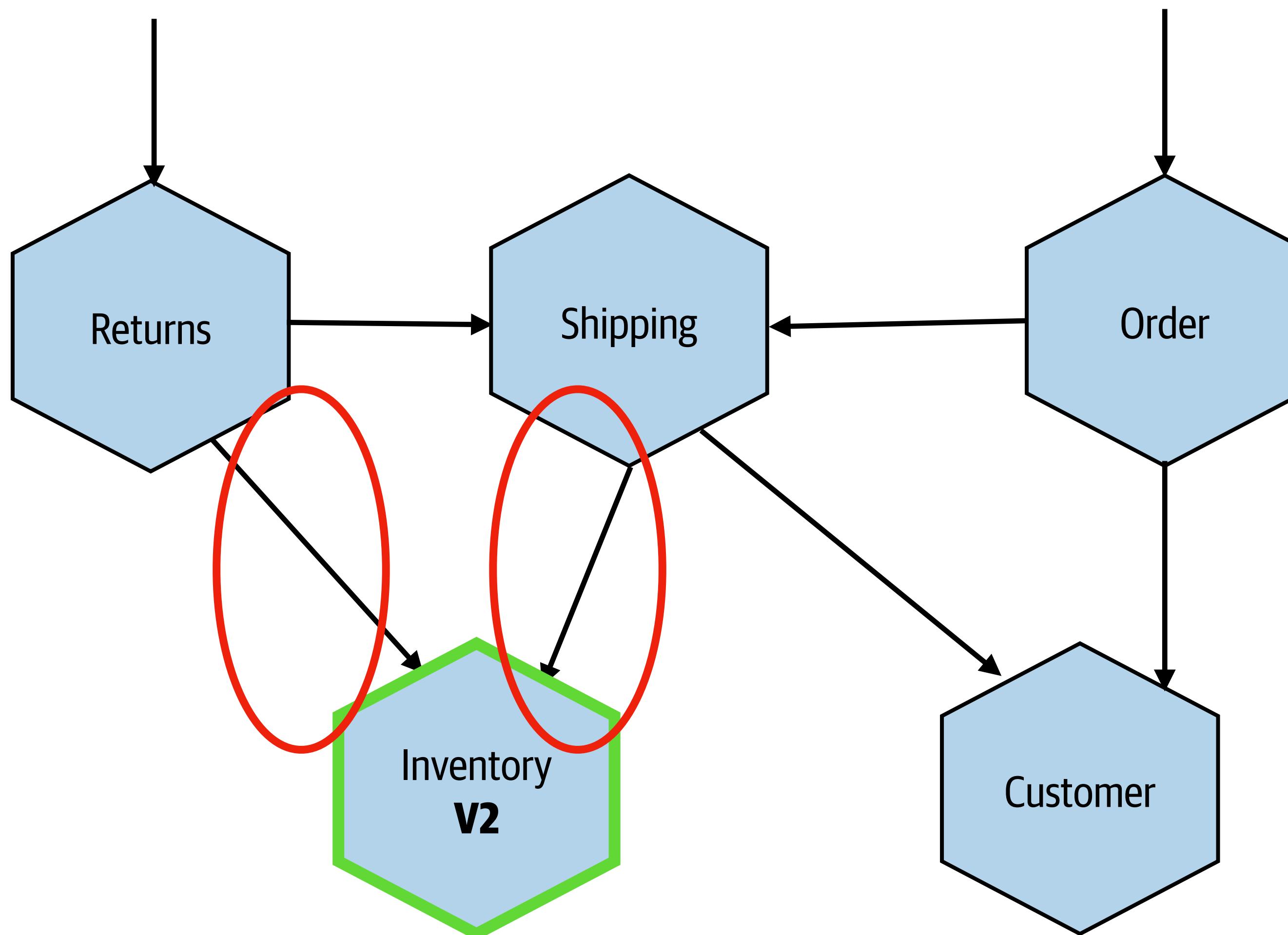
## INDEPENDENT DEPLOYABILITY



When we make a change, we need to make sure we don't break upstream consumers

So backwards compatibility is key

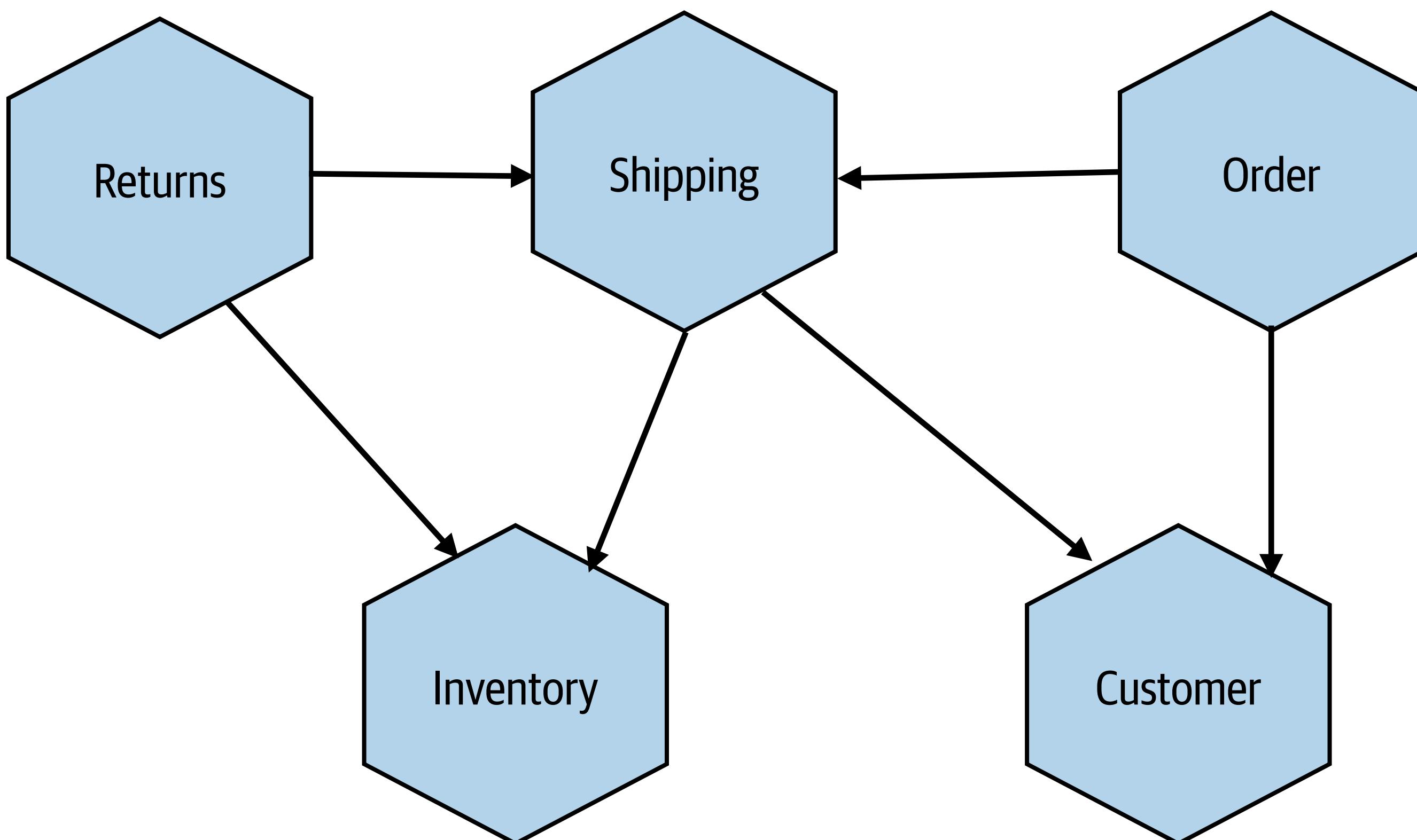
## INDEPENDENT DEPLOYABILITY



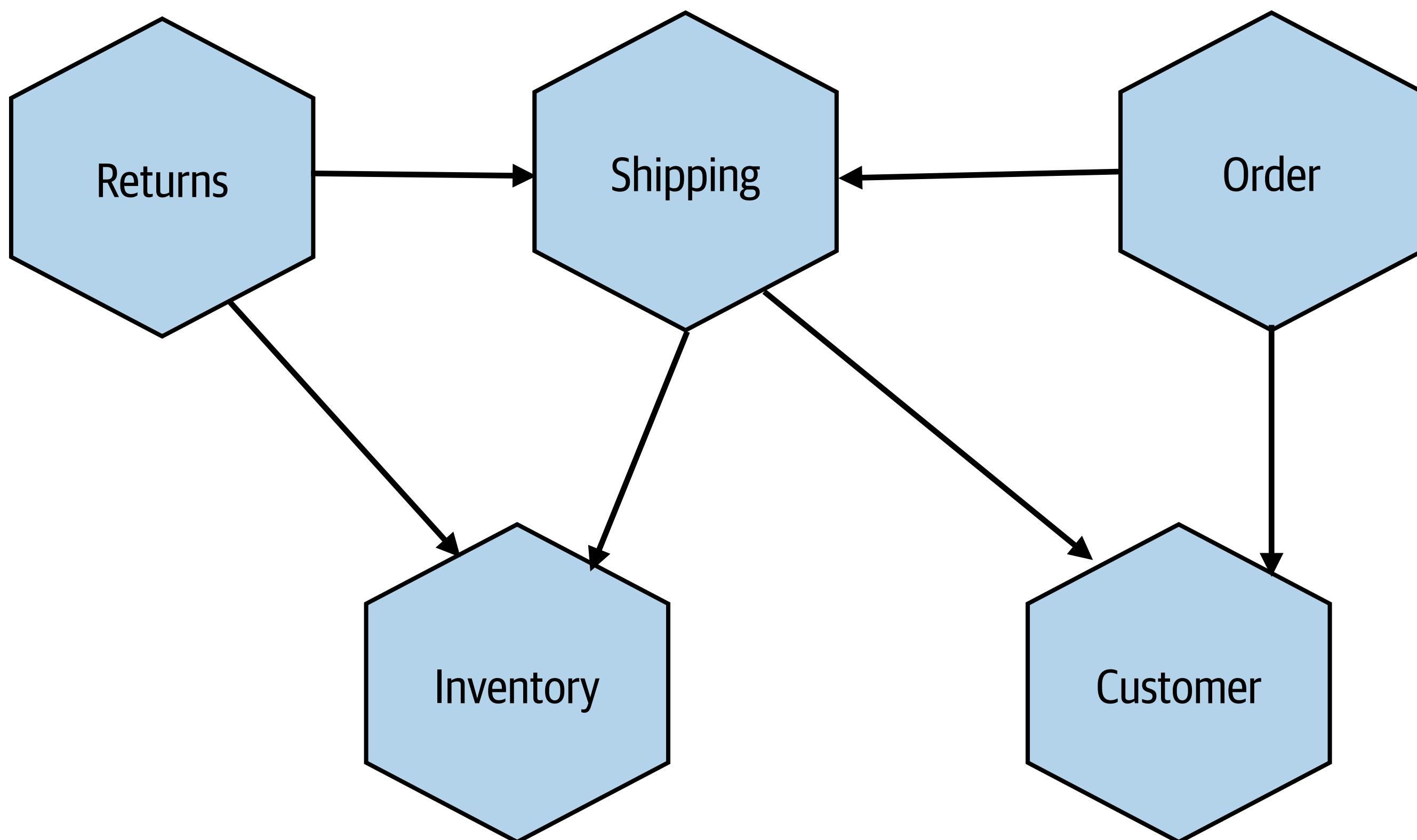
When we make a change, we need to make sure we don't break upstream consumers

So backwards compatibility is key

# DATABASES

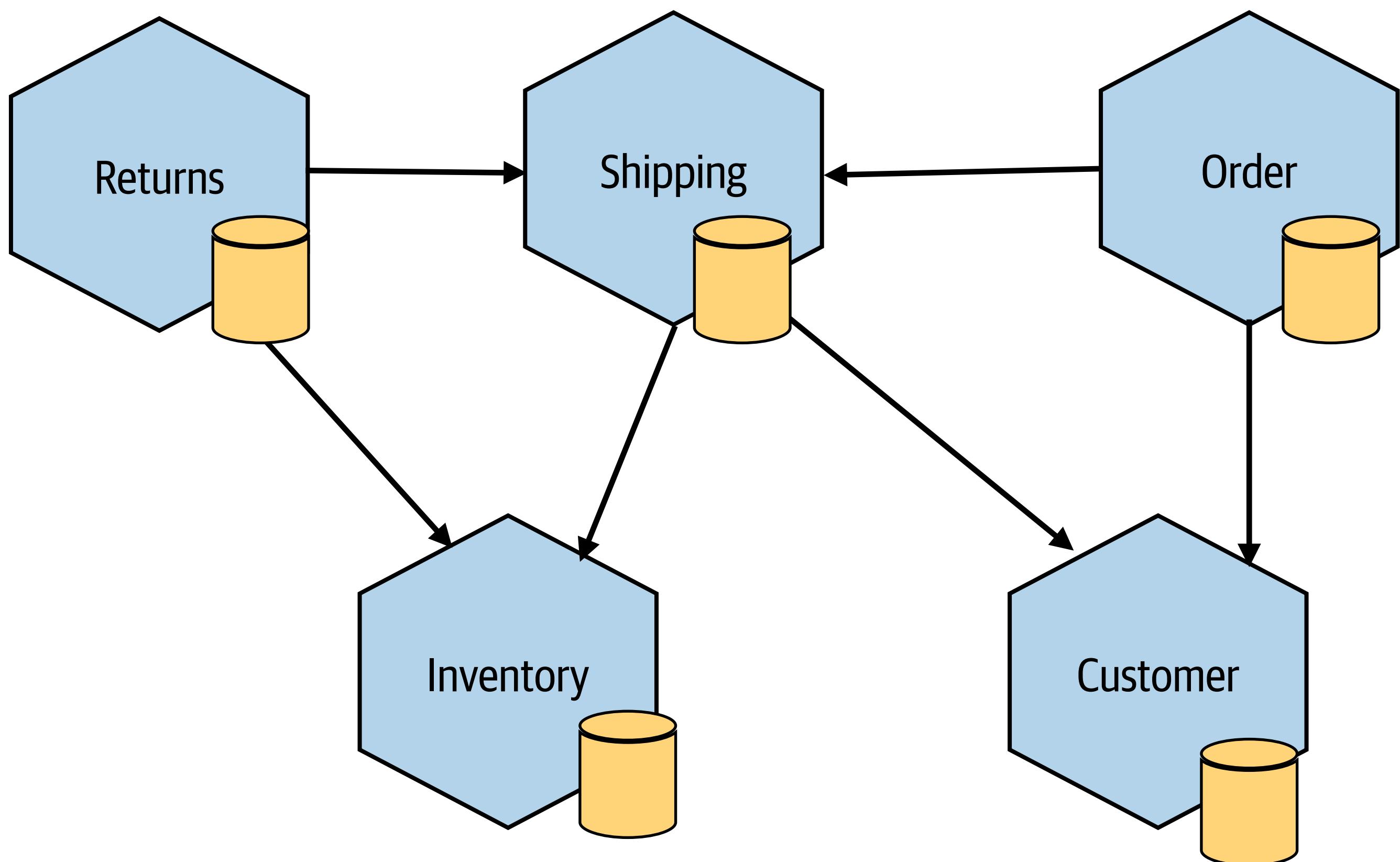


## DATABASES



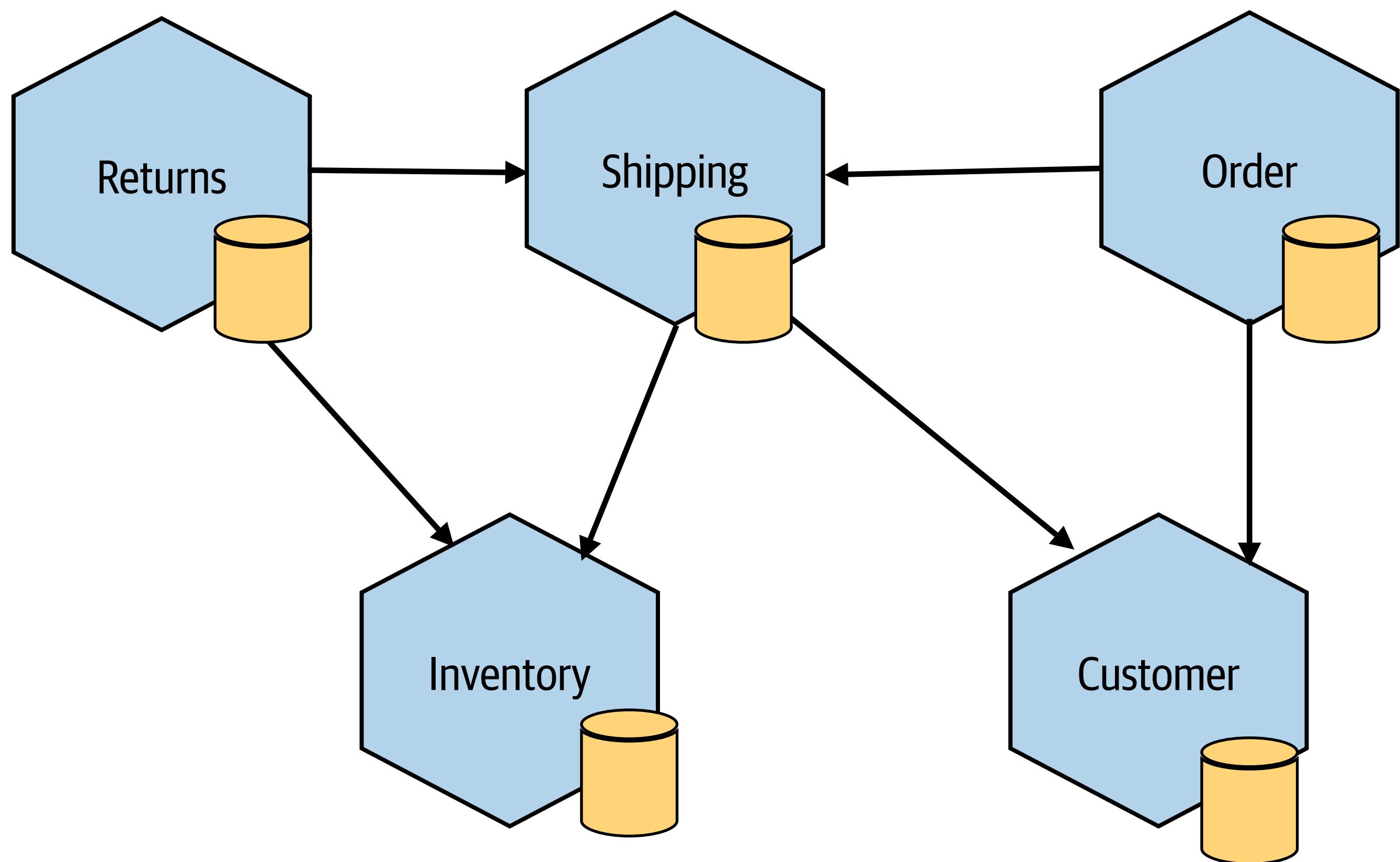
If microservices need to use a database to manage their state, this is hidden from other consumers

## DATABASES



If microservices need to use a database to manage their state, this is hidden from other consumers

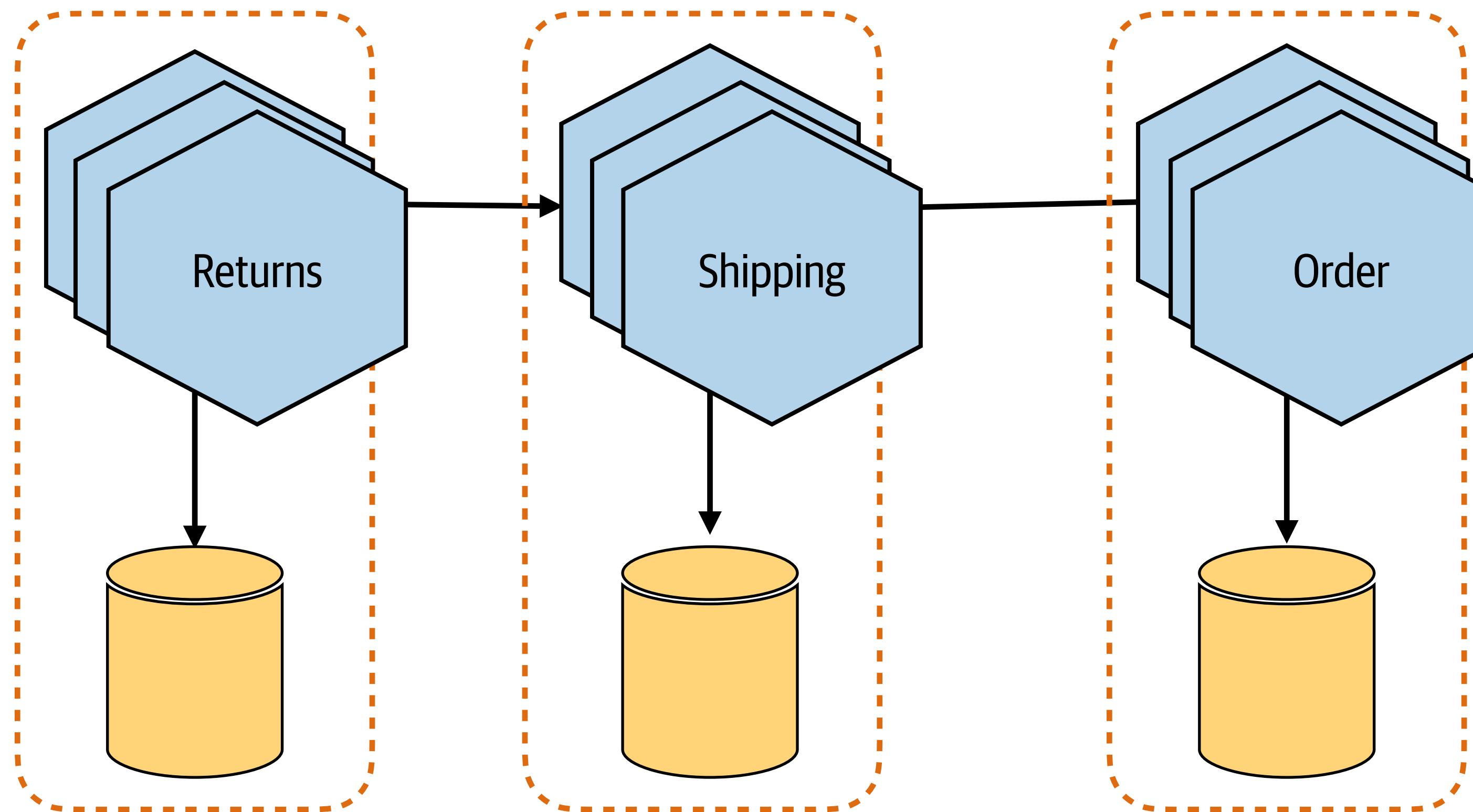
## DATABASES



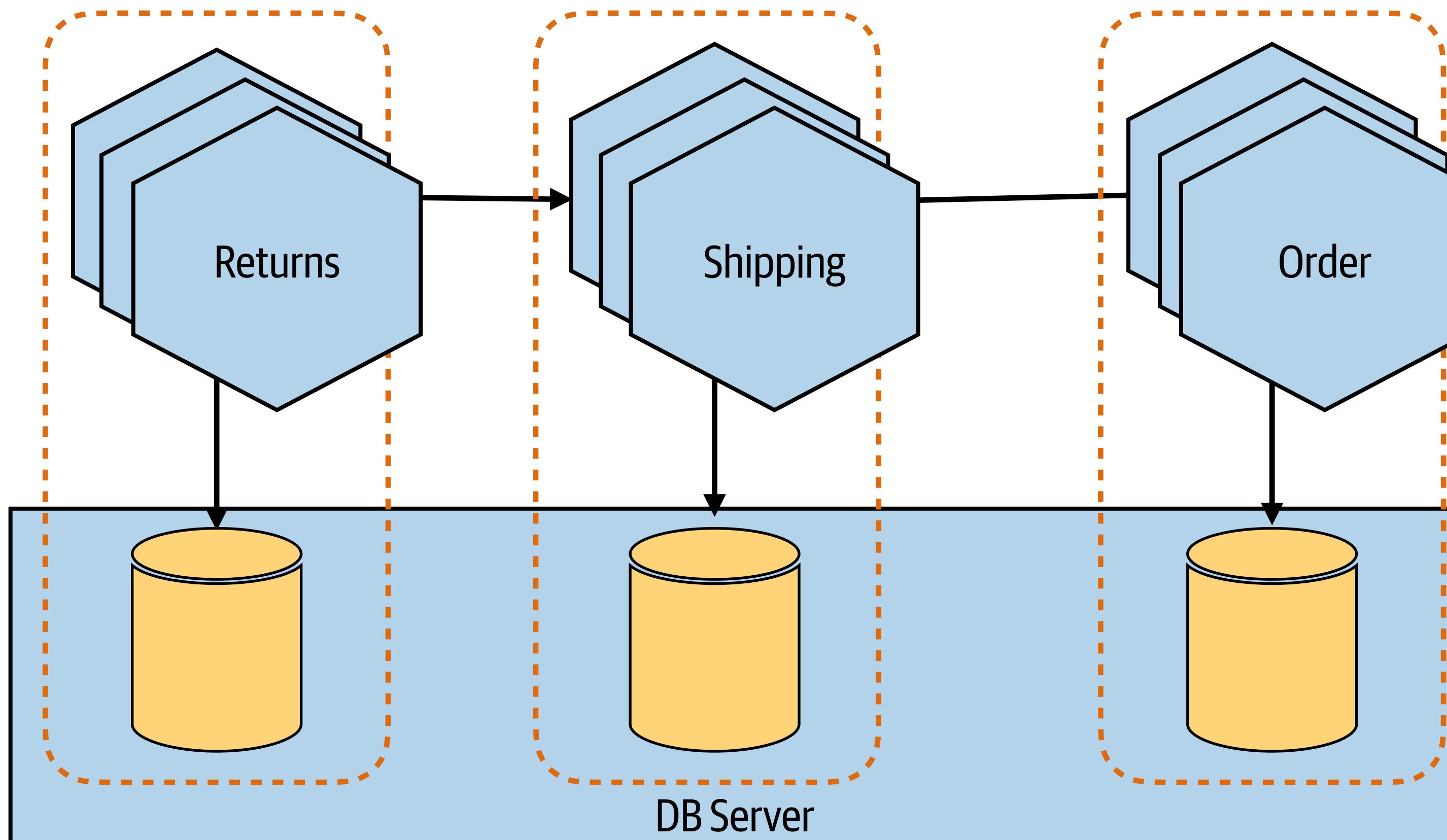
If microservices need to use a database to manage their state, this is hidden from other consumers

Does that mean lots of DB infrastructure to manage?

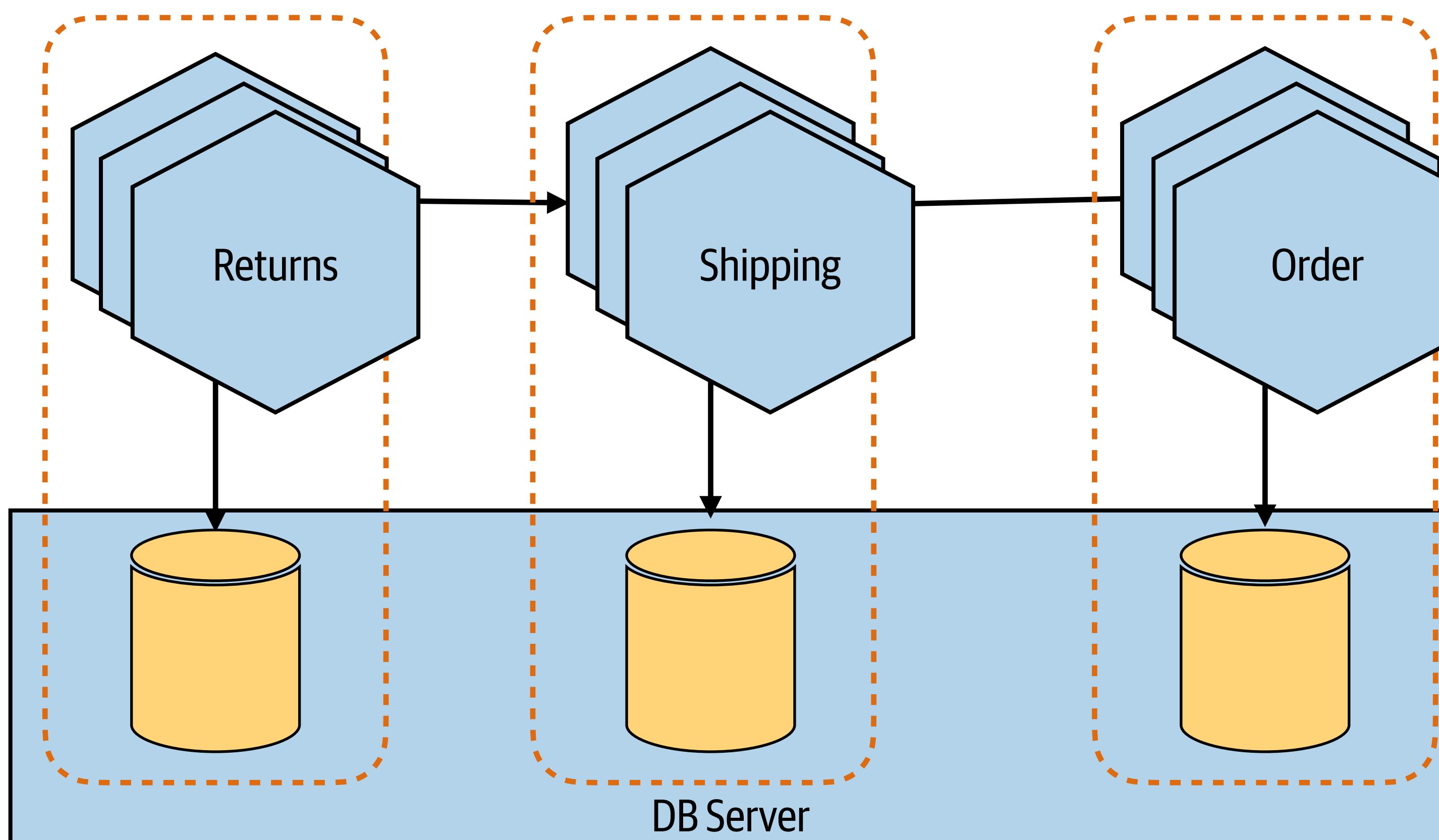
## SHARED DB INFRASTRUCTURE



## SHARED DB INFRASTRUCTURE

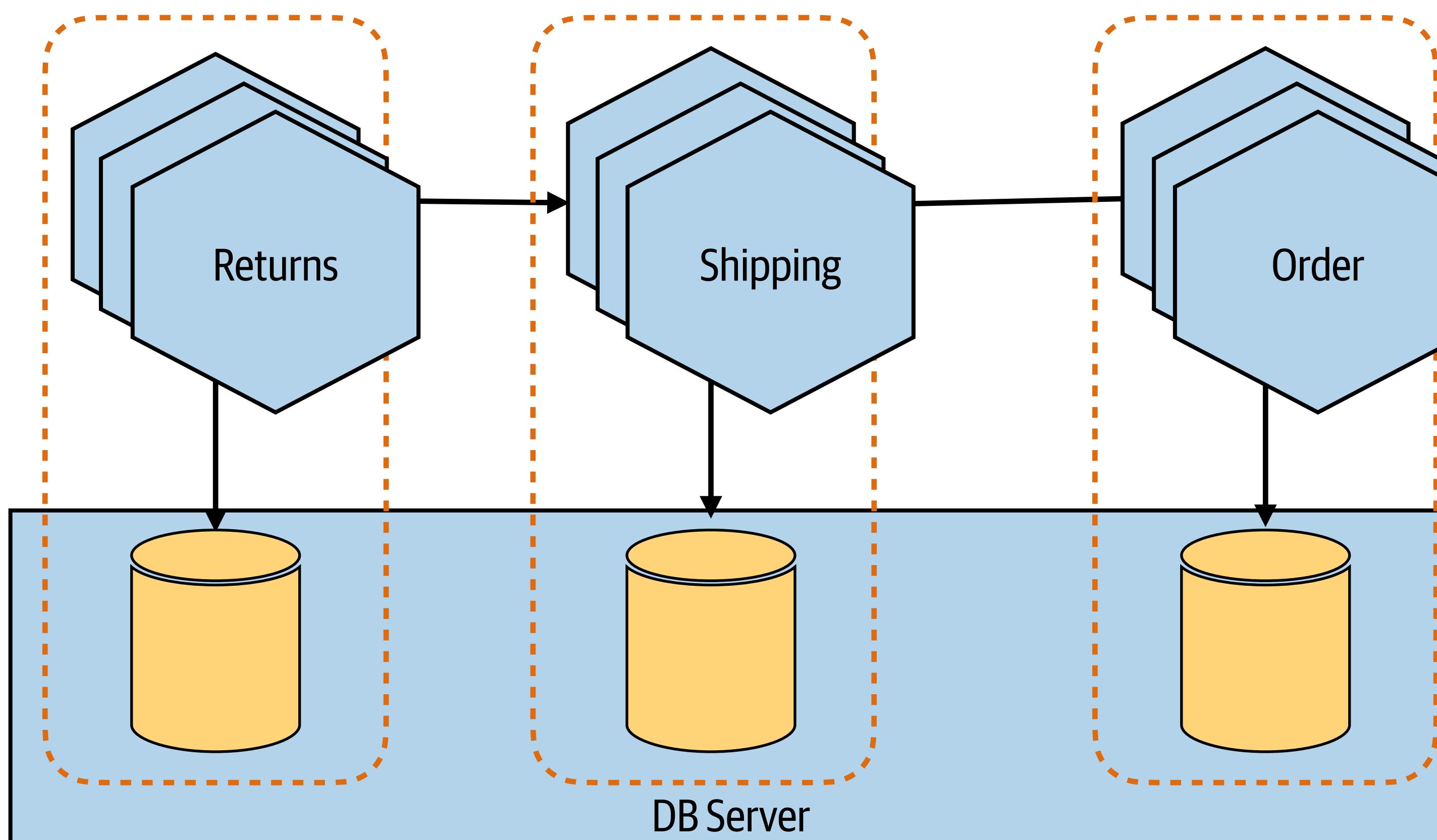


## SHARED DB INFRASTRUCTURE



Reduces the amount of DB infra and associated admin

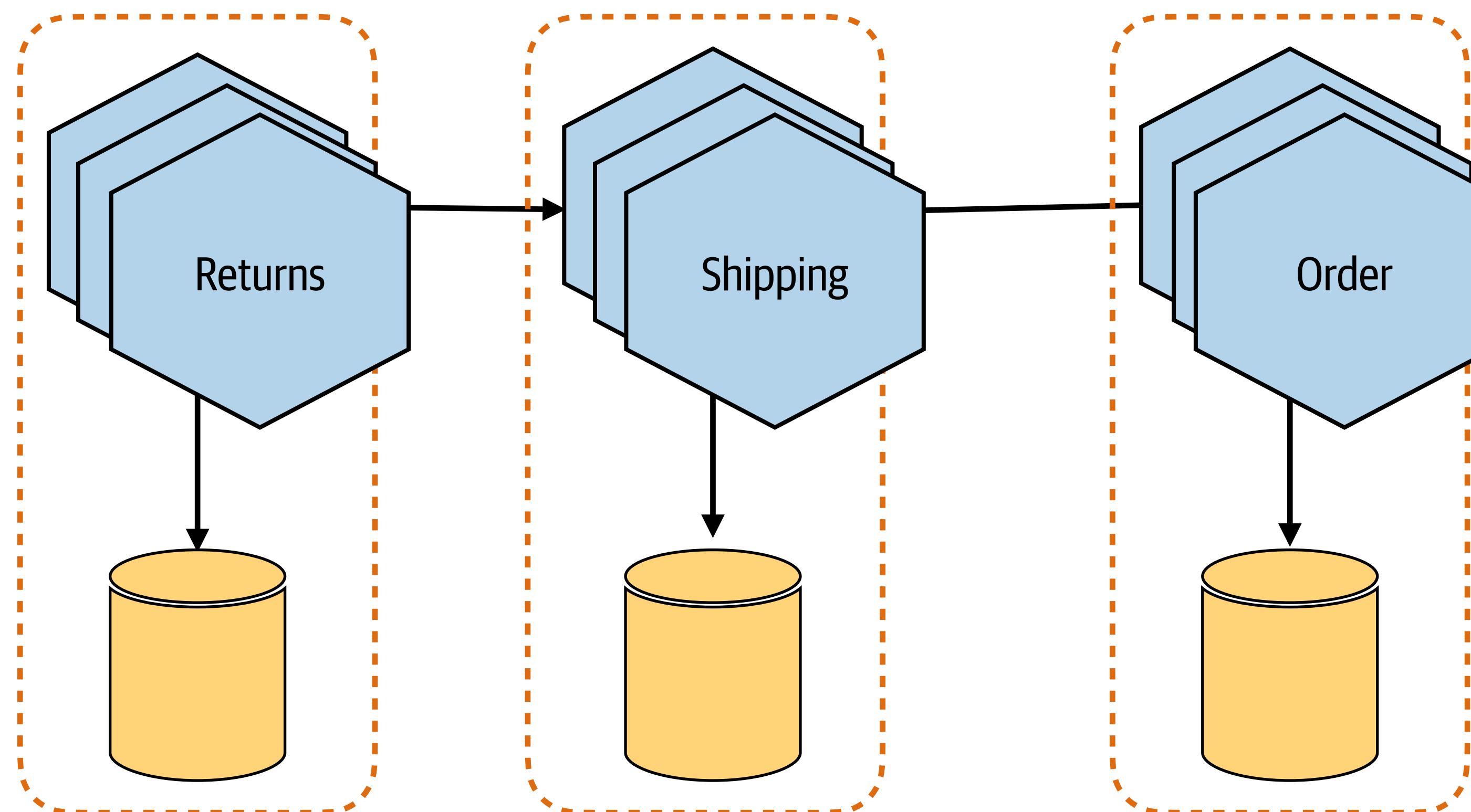
## SHARED DB INFRASTRUCTURE



Reduces the amount of DB infra and associated admin

Single source of contention

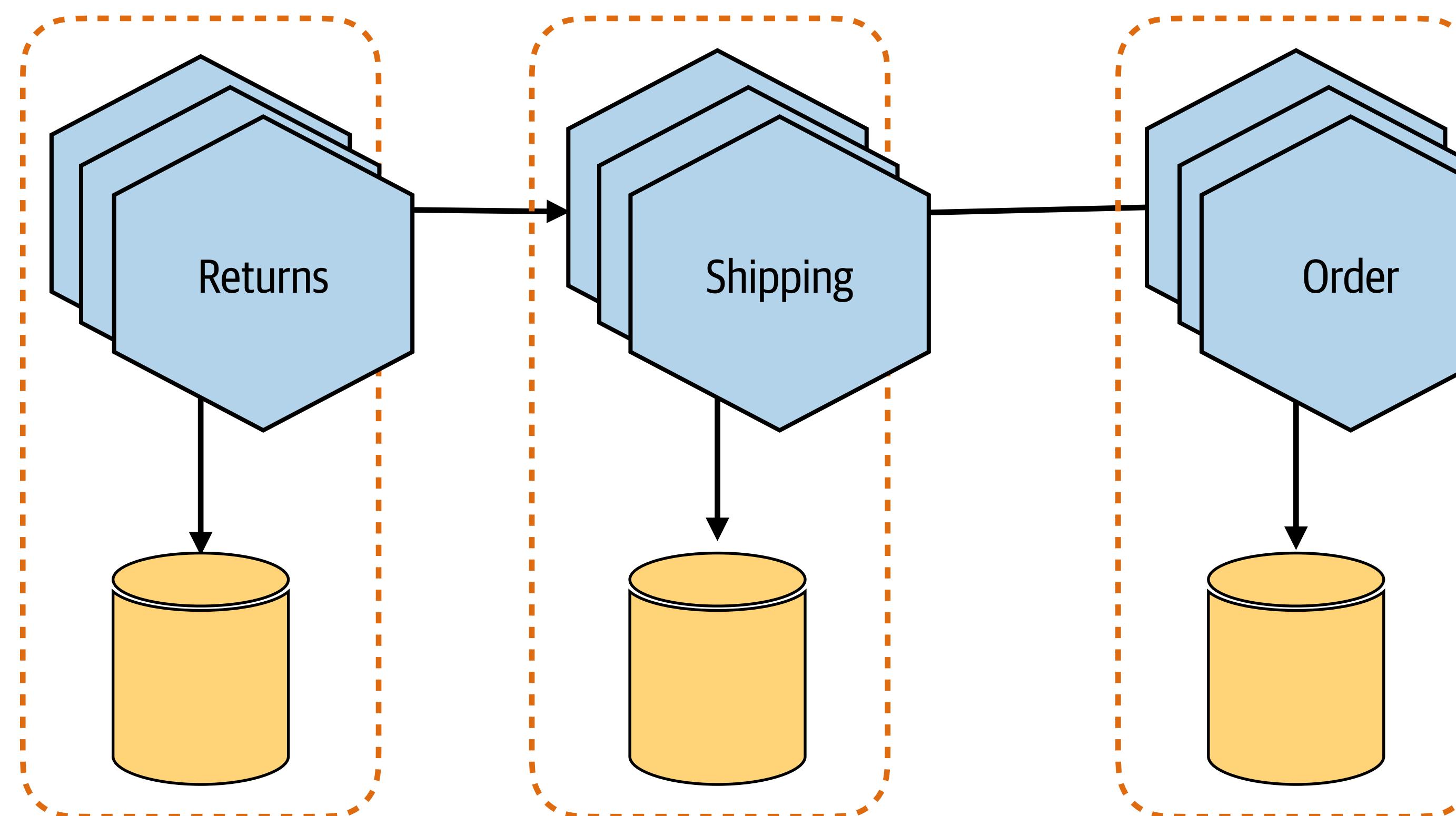
## SHARED DB INFRASTRUCTURE



Reduces the amount of DB infra and associated admin

Single source of contention

## SHARED DB INFRASTRUCTURE

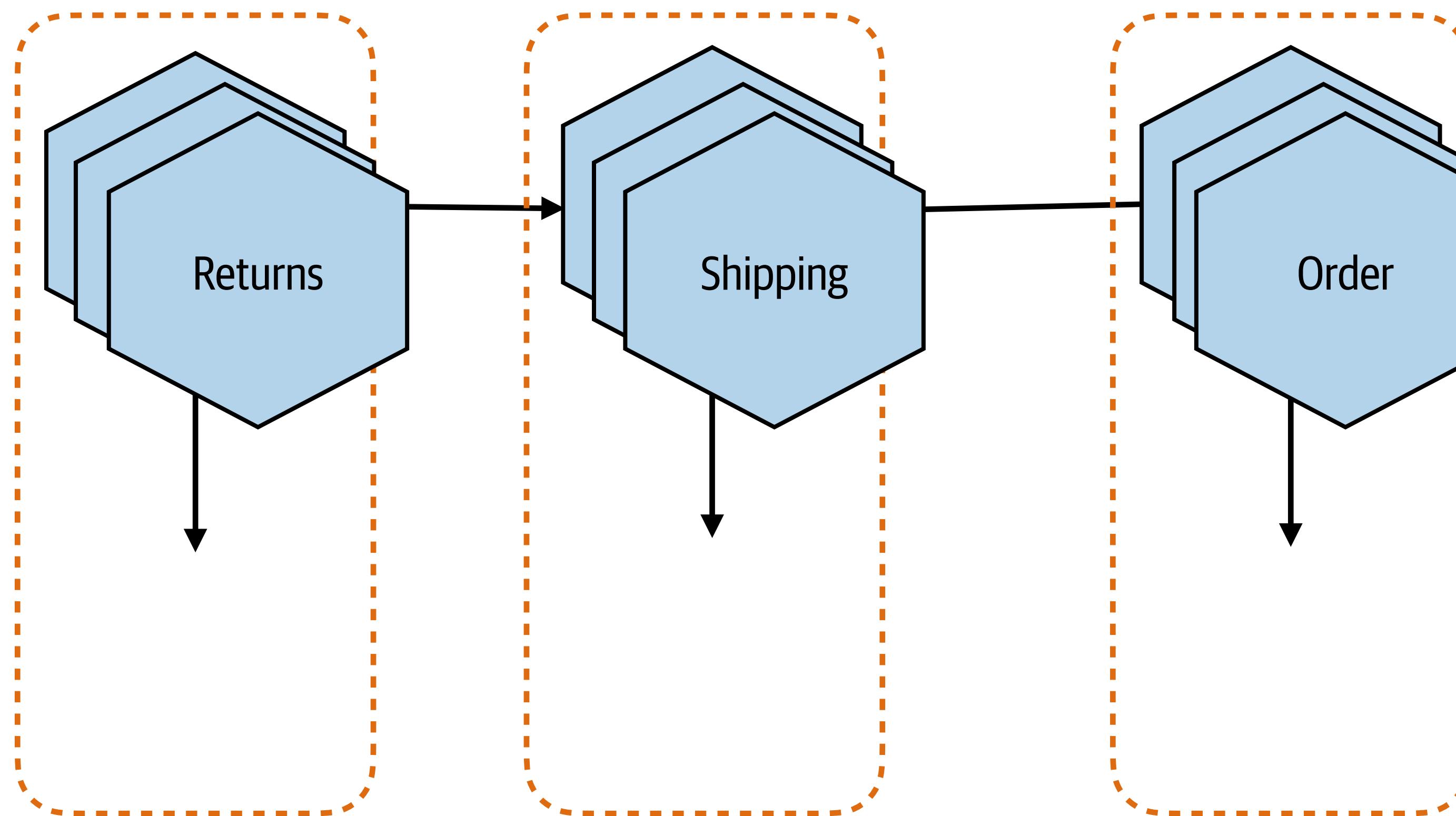


Reduces the amount of DB infra and associated admin

Single source of contention

Single source of failure

## SHARED DB INFRASTRUCTURE

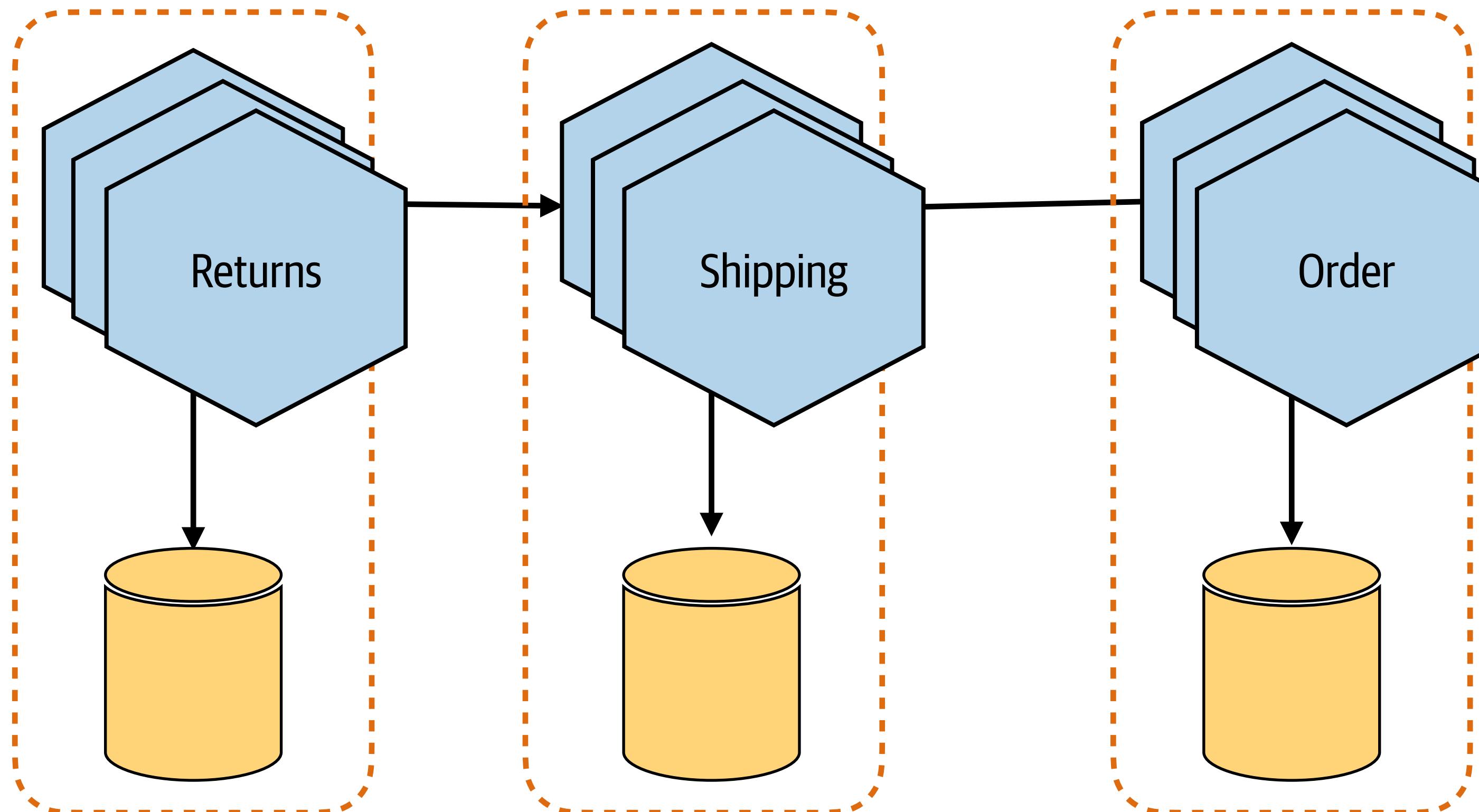


Reduces the amount of DB infra and associated admin

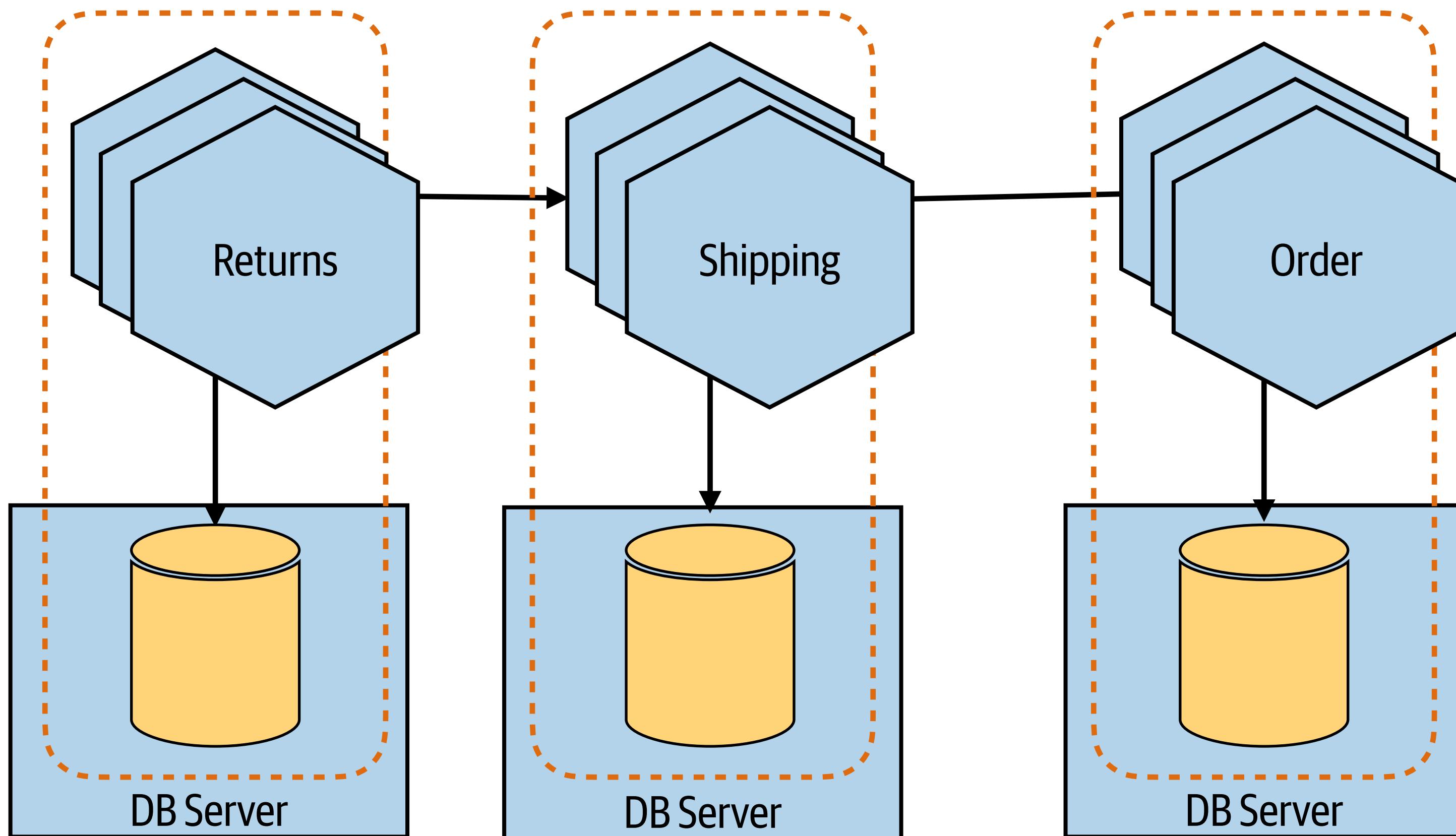
Single source of contention

Single source of failure

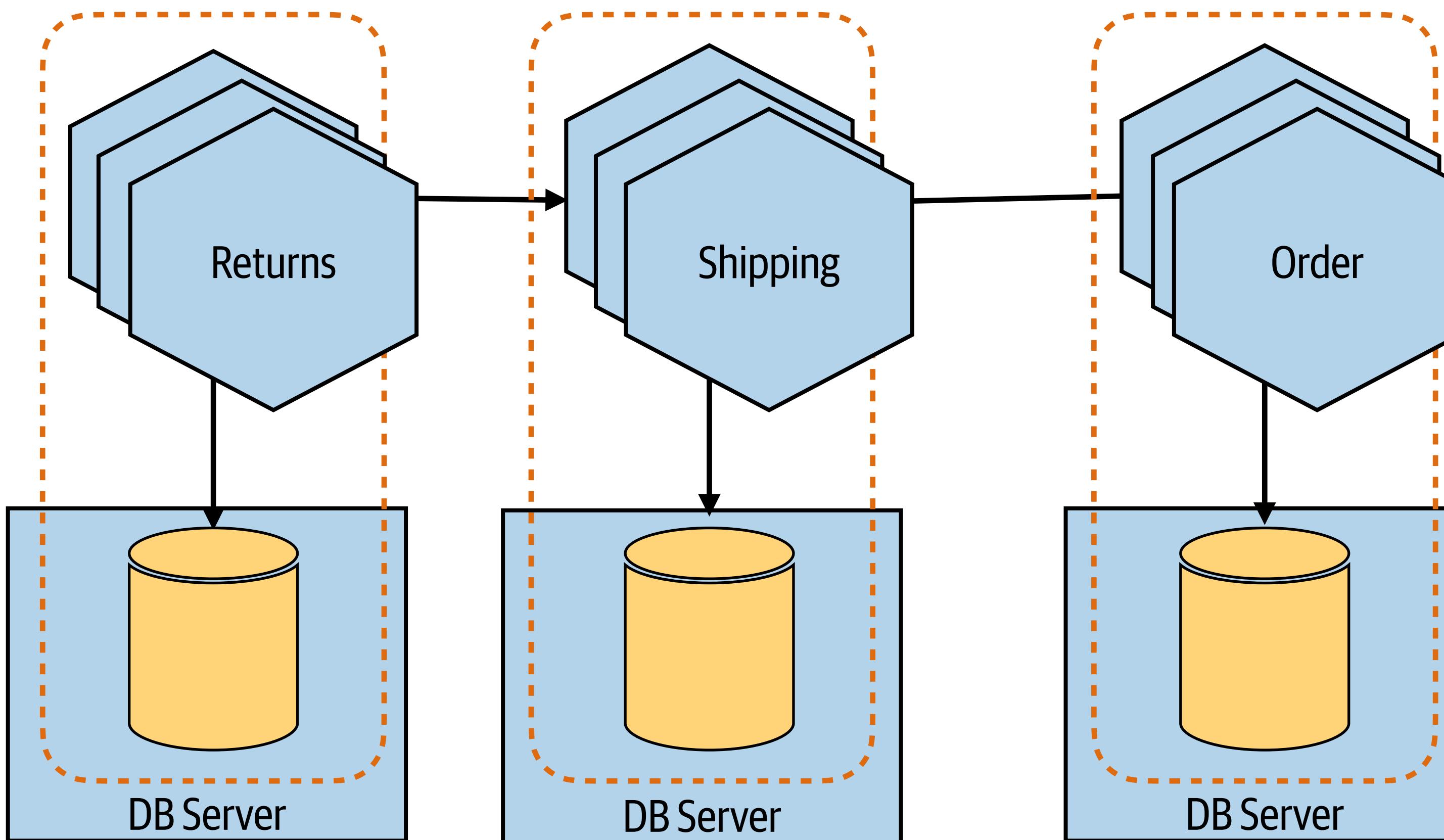
## DEDICATED DB INFRASTRUCTURE



## DEDICATED DB INFRASTRUCTURE

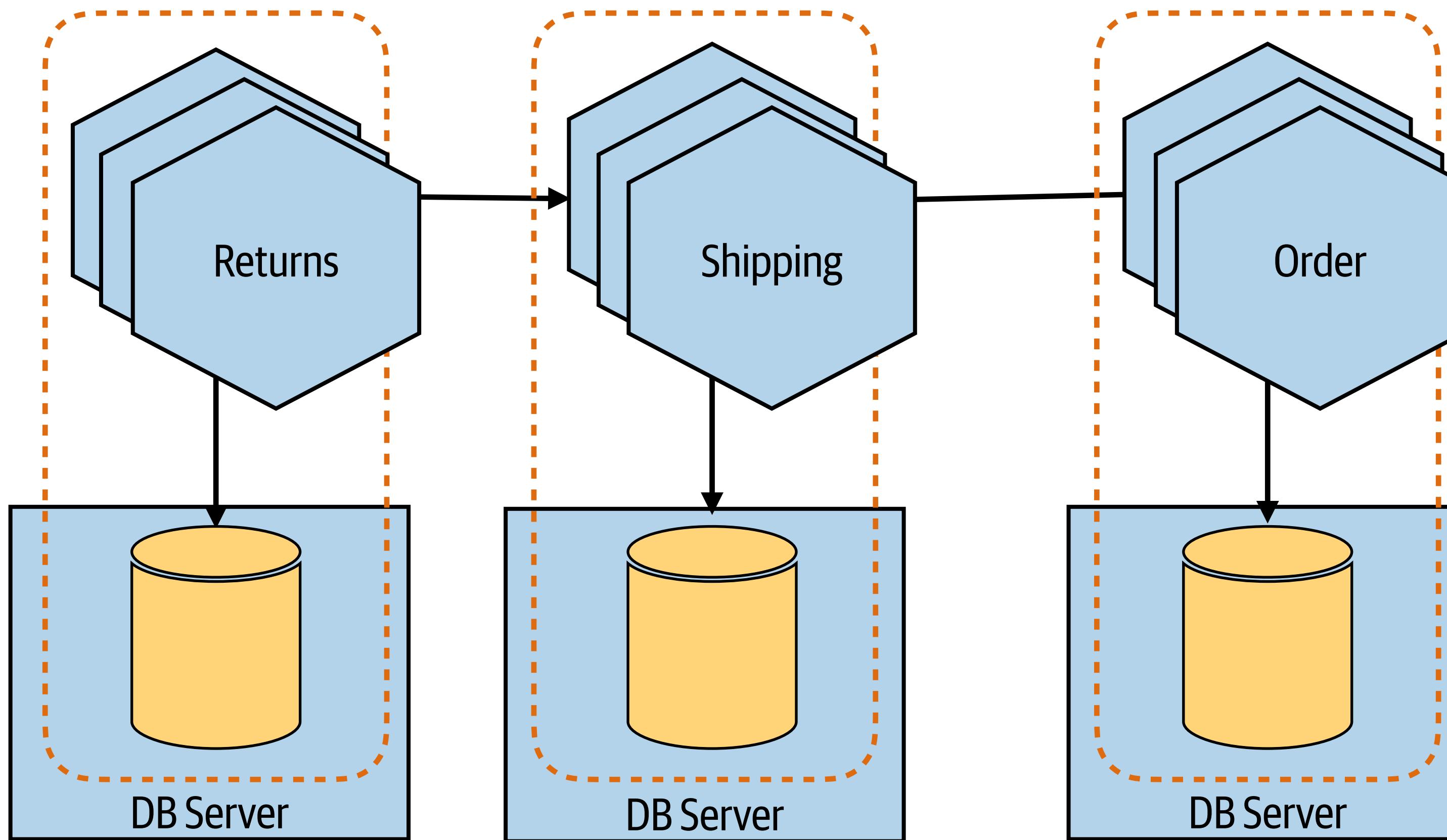


## DEDICATED DB INFRASTRUCTURE



More infra to manage

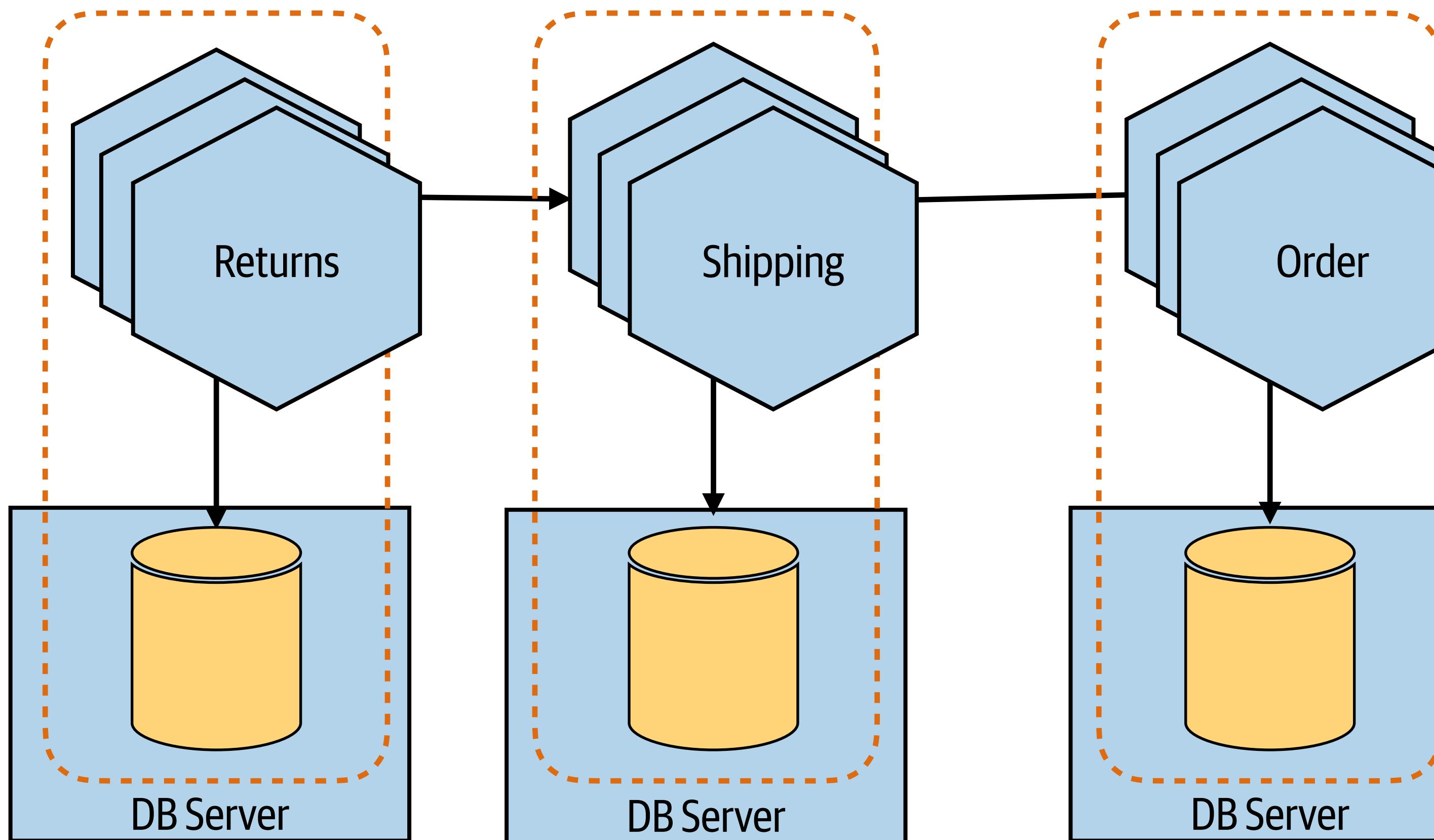
## DEDICATED DB INFRASTRUCTURE



More infra to manage

Can use different technology

## DEDICATED DB INFRASTRUCTURE

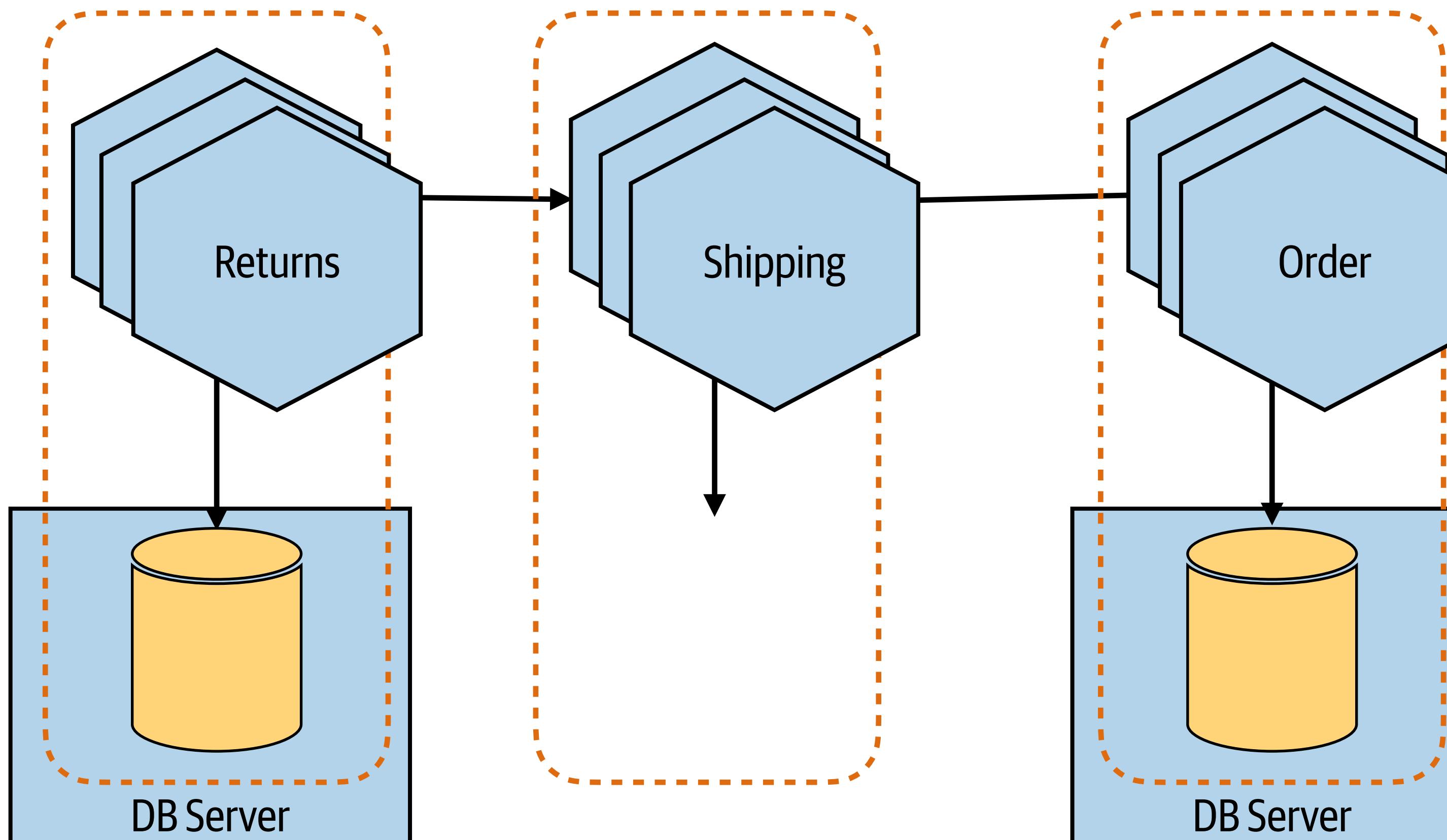


More infra to manage

Can use different technology

Avoids contention  
and reduces blast  
radius of any failure

## DEDICATED DB INFRASTRUCTURE



More infra to manage

Can use different technology

Avoids contention  
and reduces blast  
radius of any failure

**POLL: WHAT DATABASES DO YOU USE? SELECT ALL THAT APPLY  
(ALLOW ATTENDEE TO SELECT MULTIPLE)**

Relational (e.g. mysql, Oracle)

Document (e.g. Cosmos, dynamo, Mongo)

Graph (e.g. Neo4j)

Column (e.g. Cassandra)

Key/Value (e.g. Riak)

Other (please specify)

## GENERAL OBSERVATIONS

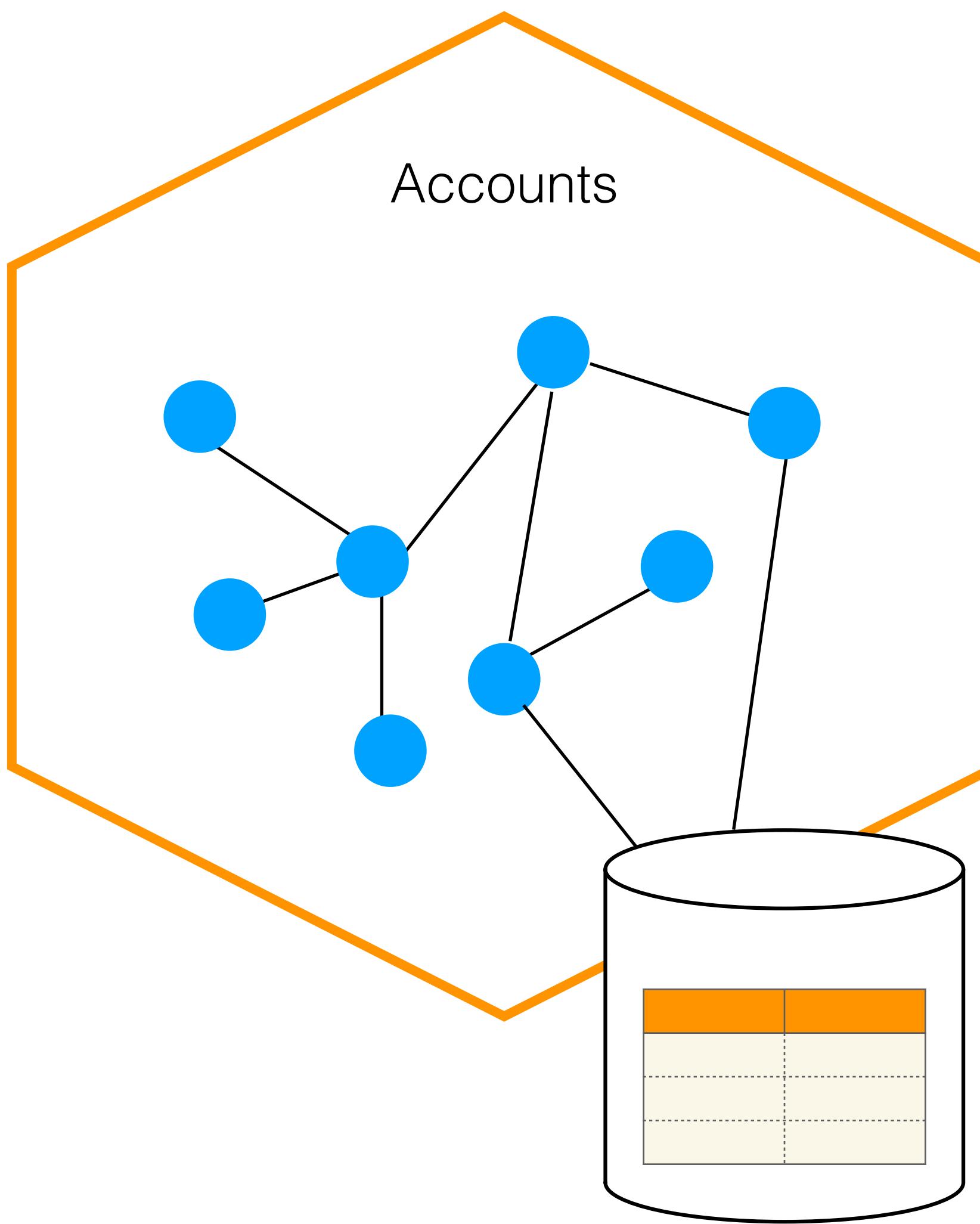
When managing your own infrastructure  
(traditional DC, private cloud), then Shared  
DB Infrastructure is most common

## GENERAL OBSERVATIONS

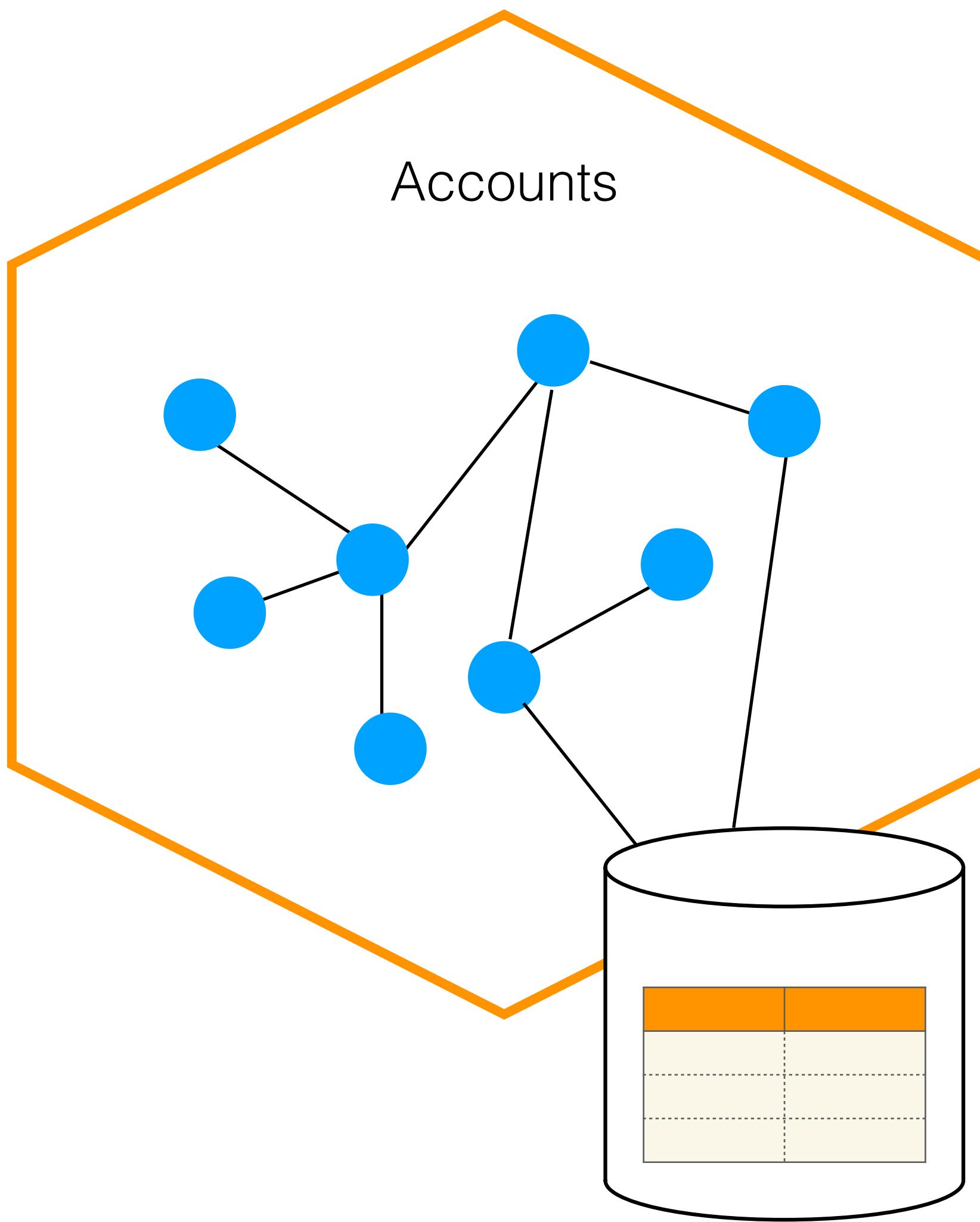
When managing your own infrastructure  
(traditional DC, private cloud), then **Shared DB Infrastructure** is most common

**Isolated DB Infrastructure** is much easier  
to justify on public cloud

## NOT HIDING?



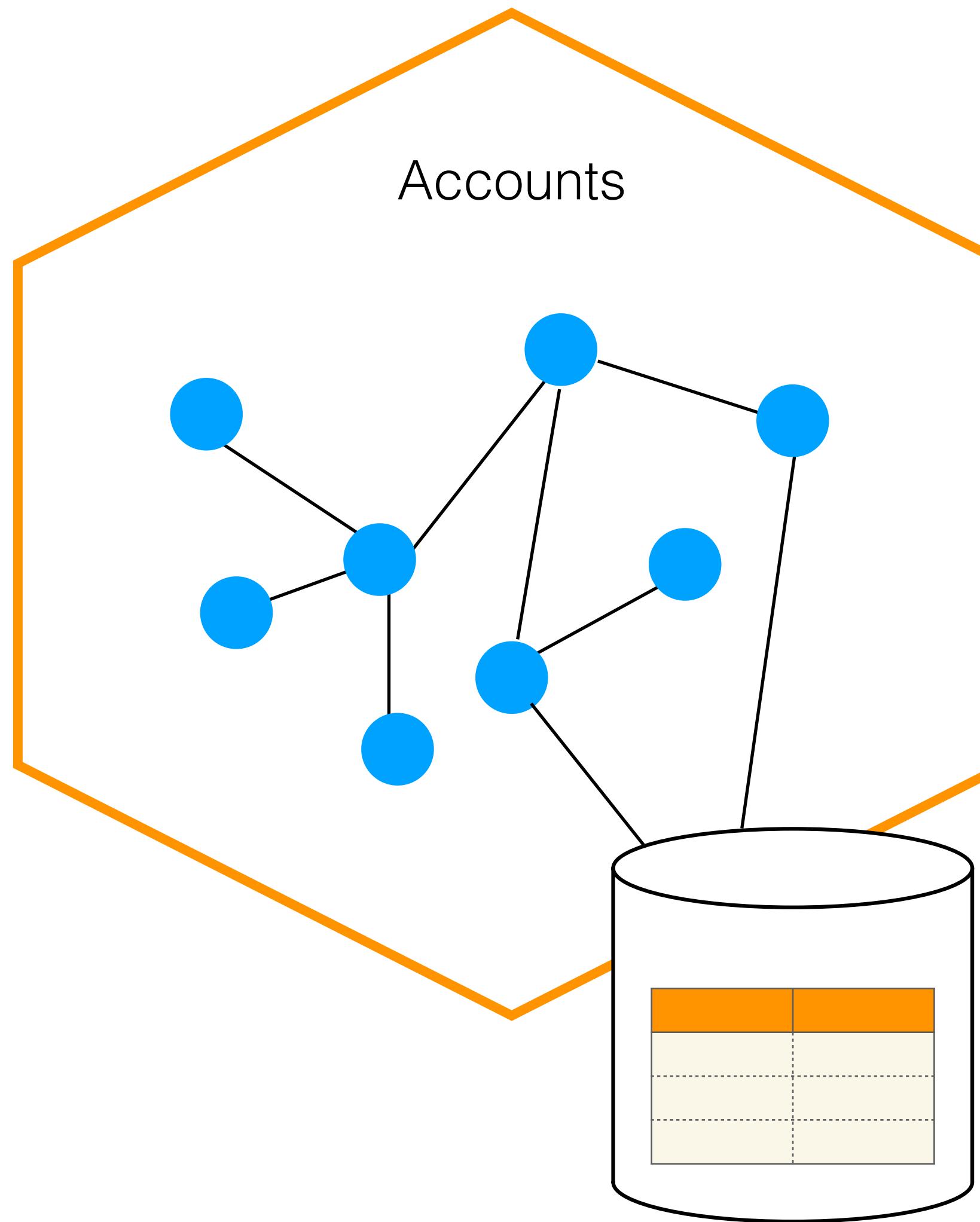
## NOT HIDING?



If an upstream consumer  
can reach into your internal  
implementation..

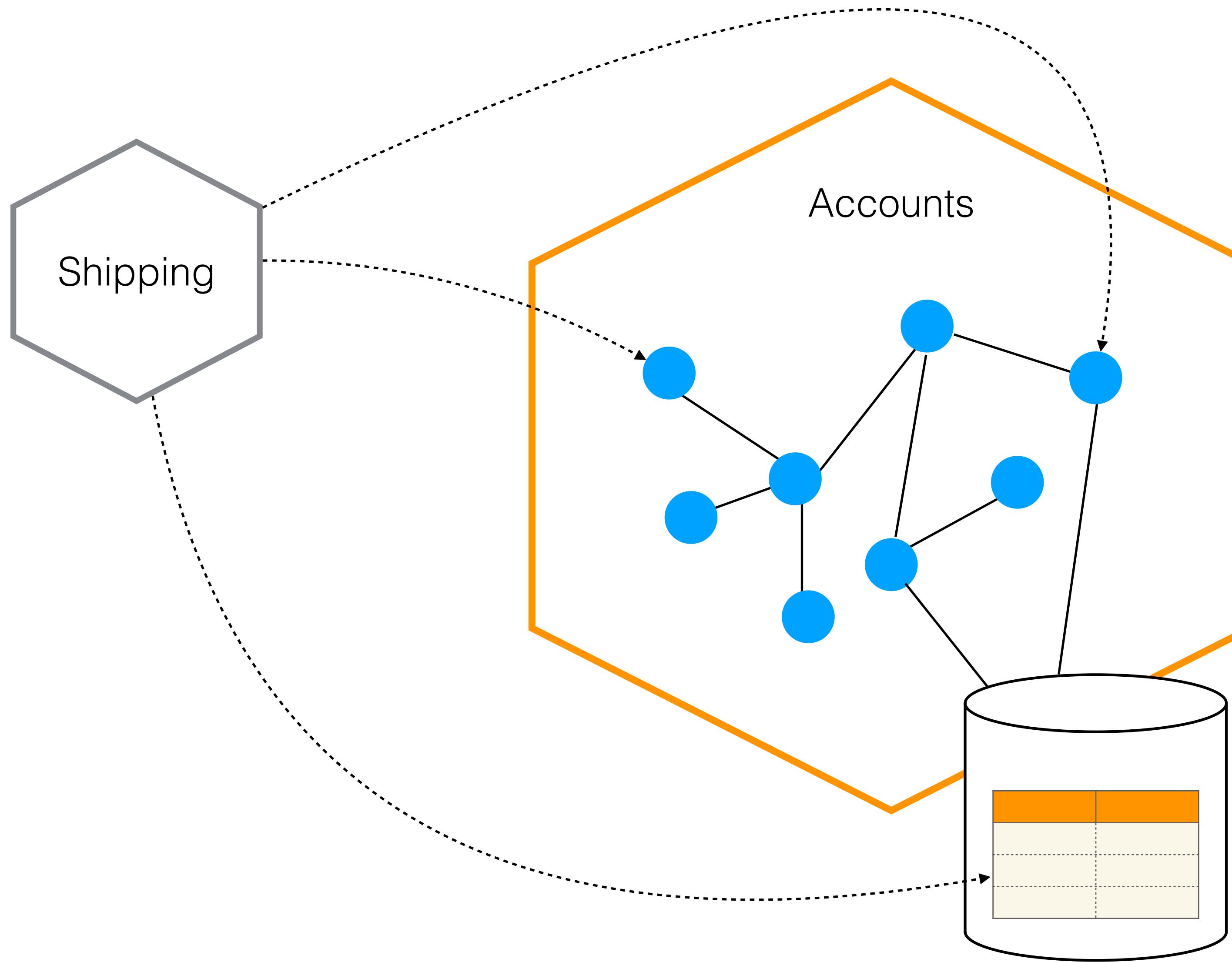
## NOT HIDING?

Shipping



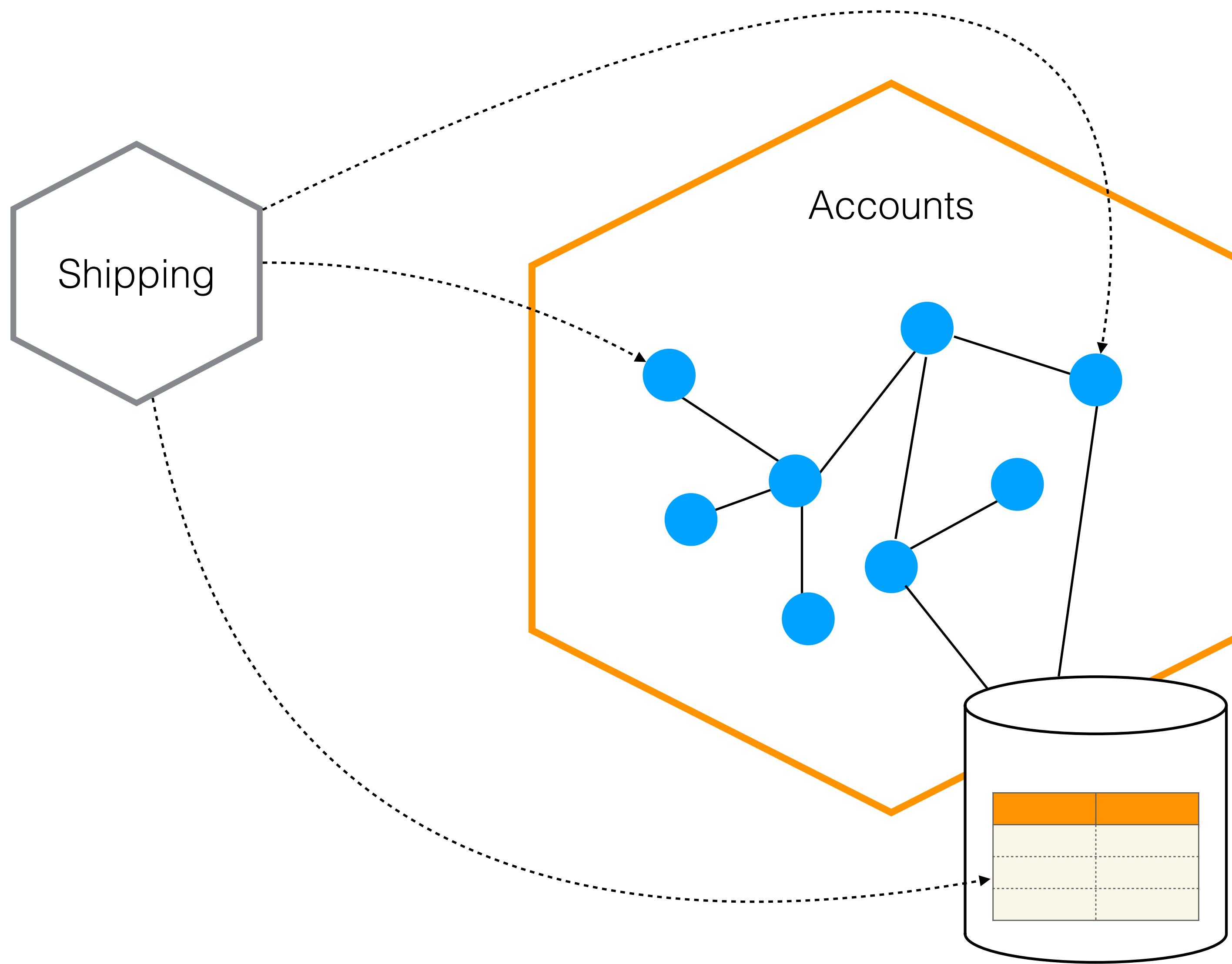
If an upstream consumer  
can reach into your internal  
implementation..

## NOT HIDING?



If an upstream consumer  
can reach into your internal  
implementation..

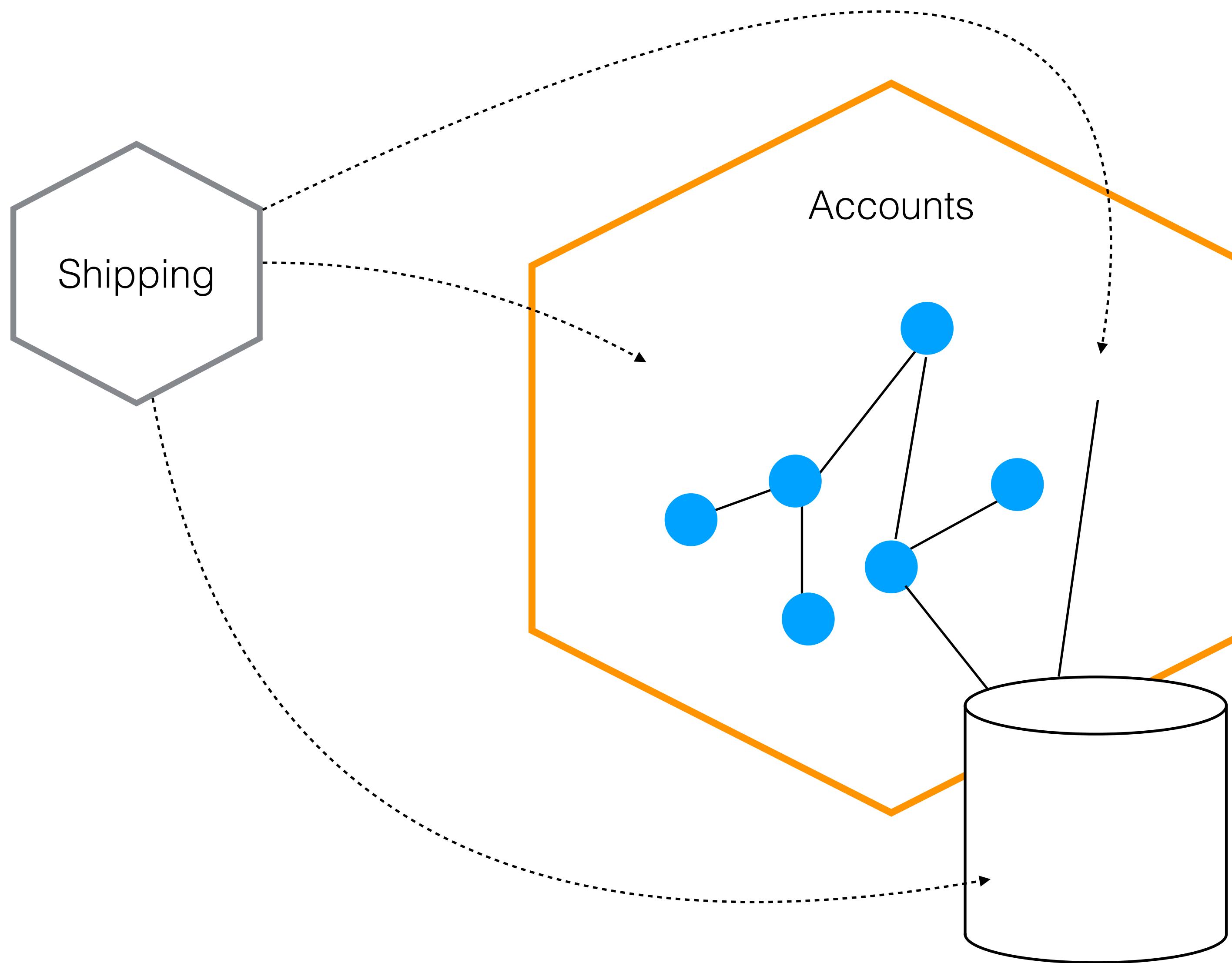
## NOT HIDING?



If an upstream consumer  
can reach into your internal  
implementation..

...then you can't change  
this implementation without  
breaking the consumer

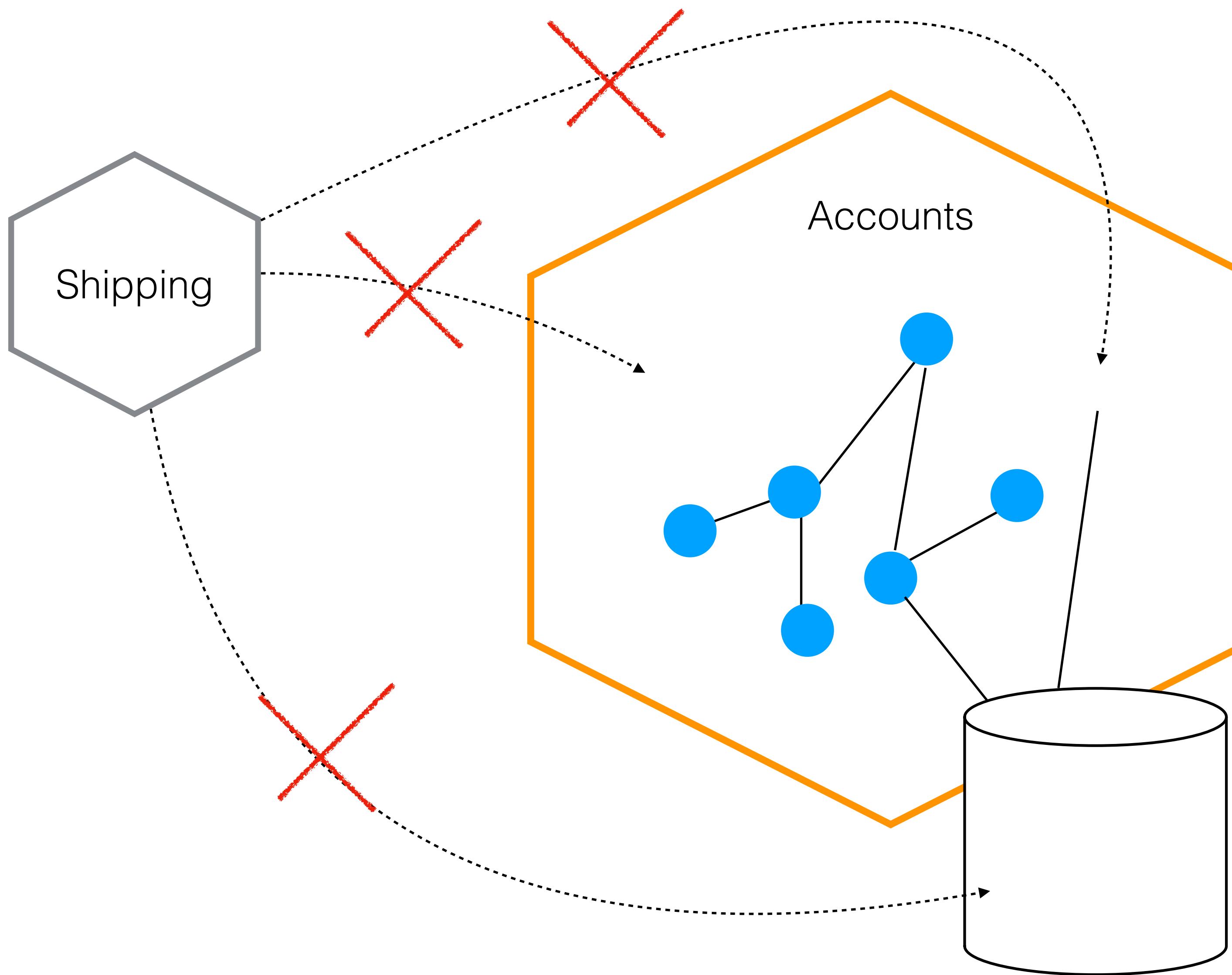
## NOT HIDING?



If an upstream consumer  
can reach into your internal  
implementation..

...then you can't change  
this implementation without  
breaking the consumer

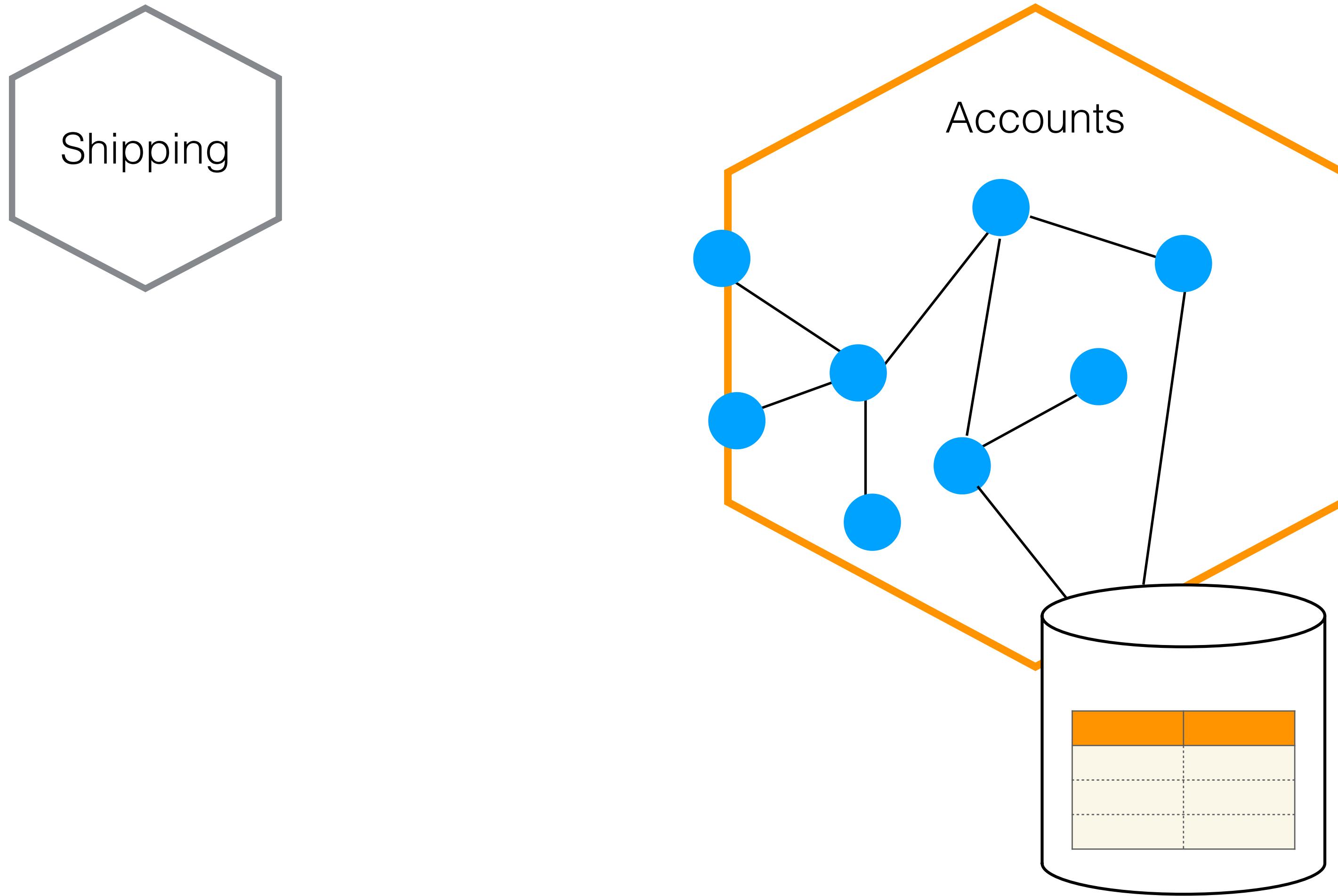
## NOT HIDING?



If an upstream consumer  
can reach into your internal  
implementation..

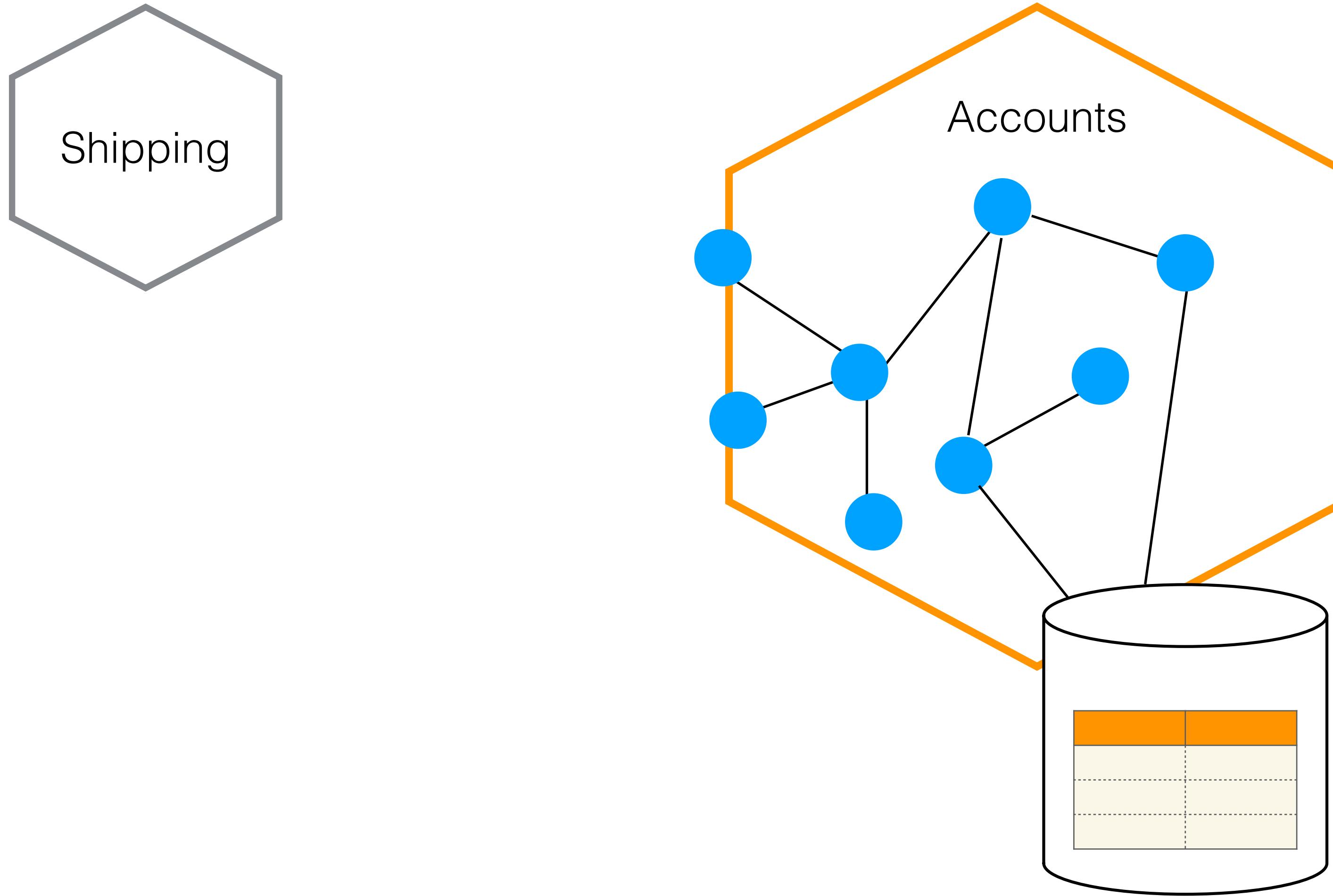
...then you can't change  
this implementation without  
breaking the consumer

# HIDING!

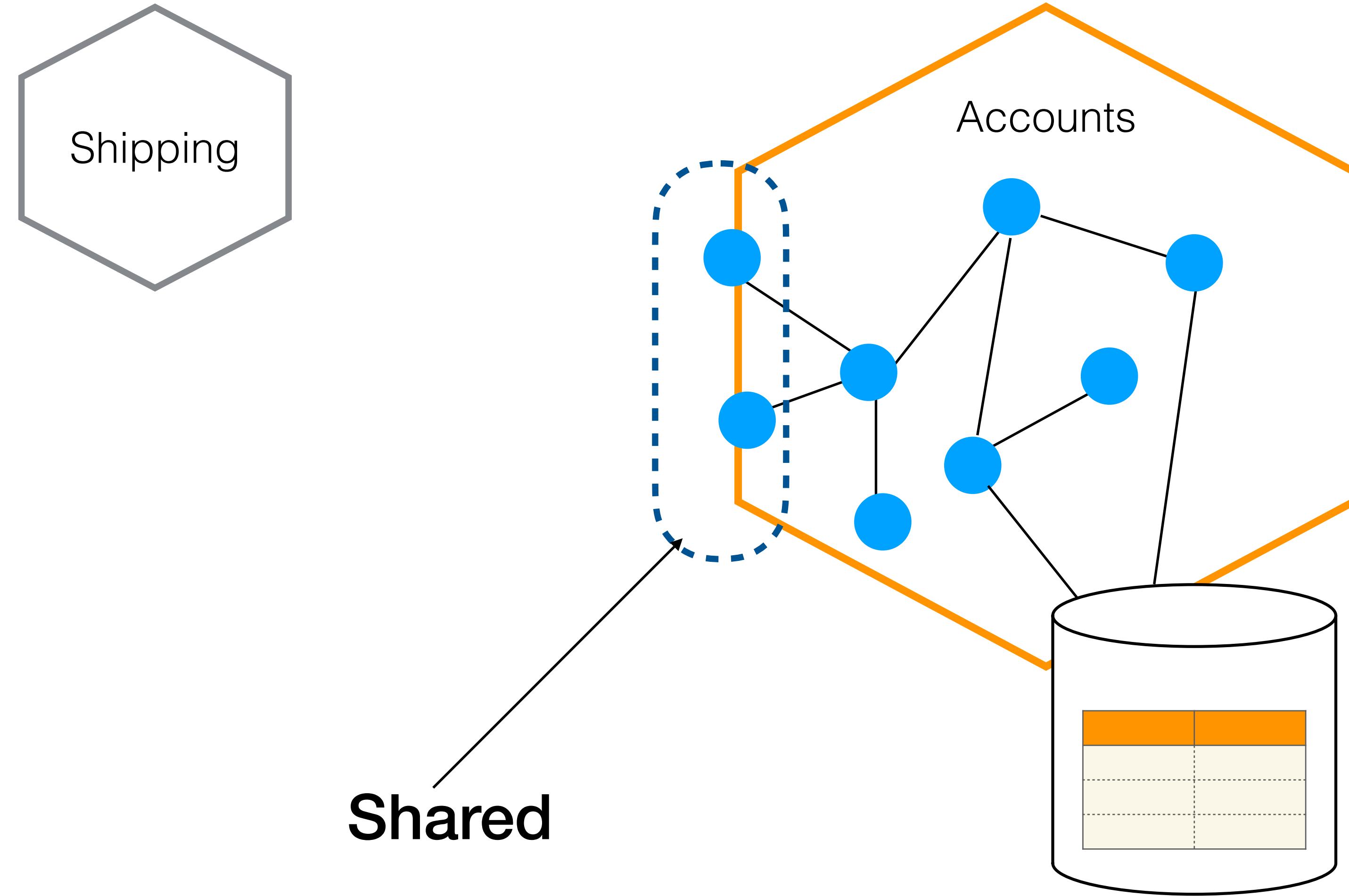


**HIDING!**

**Hide your secrets!**



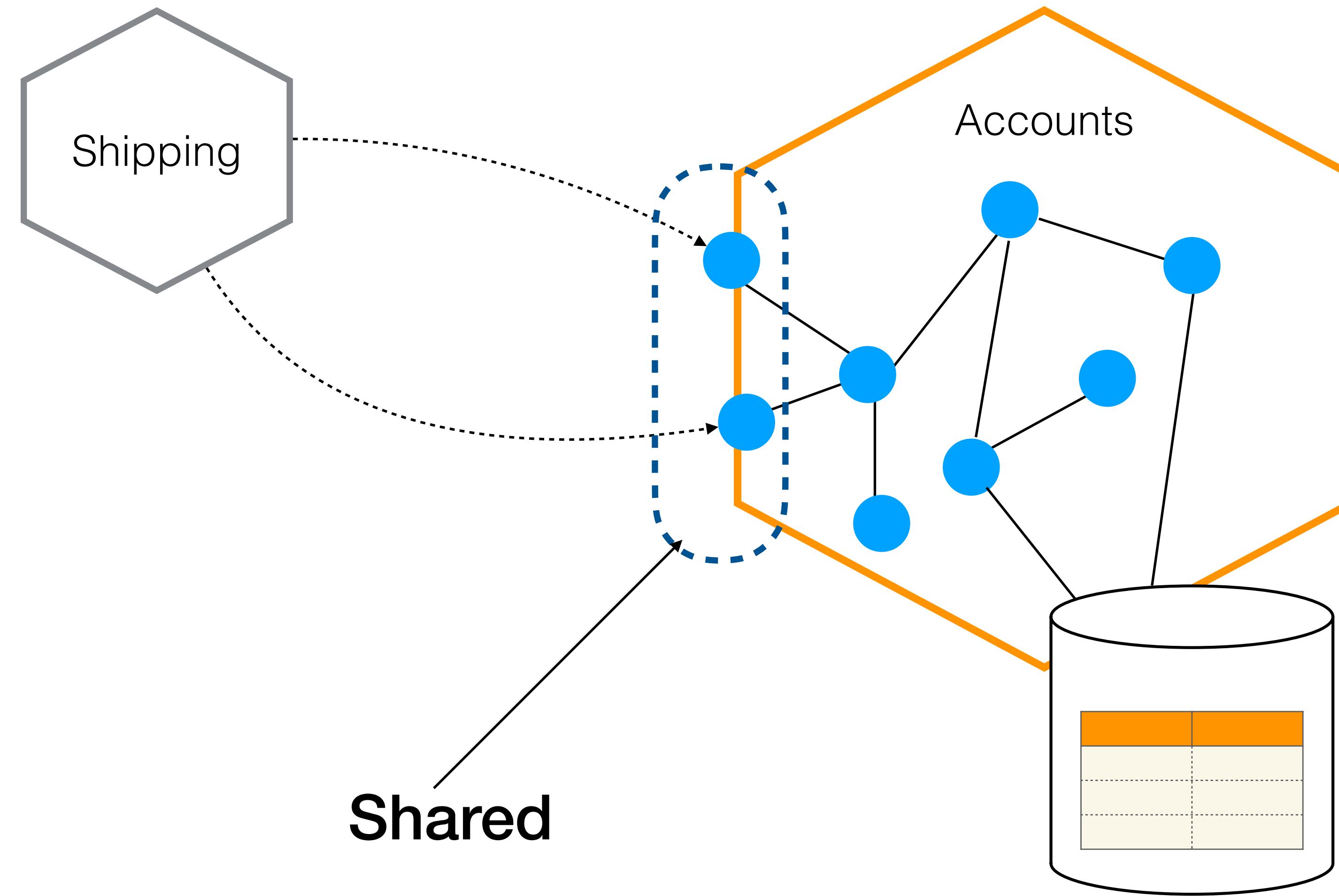
# HIDING!



**Hide your secrets!**

**Be explicit about what is shared, and what is hidden**

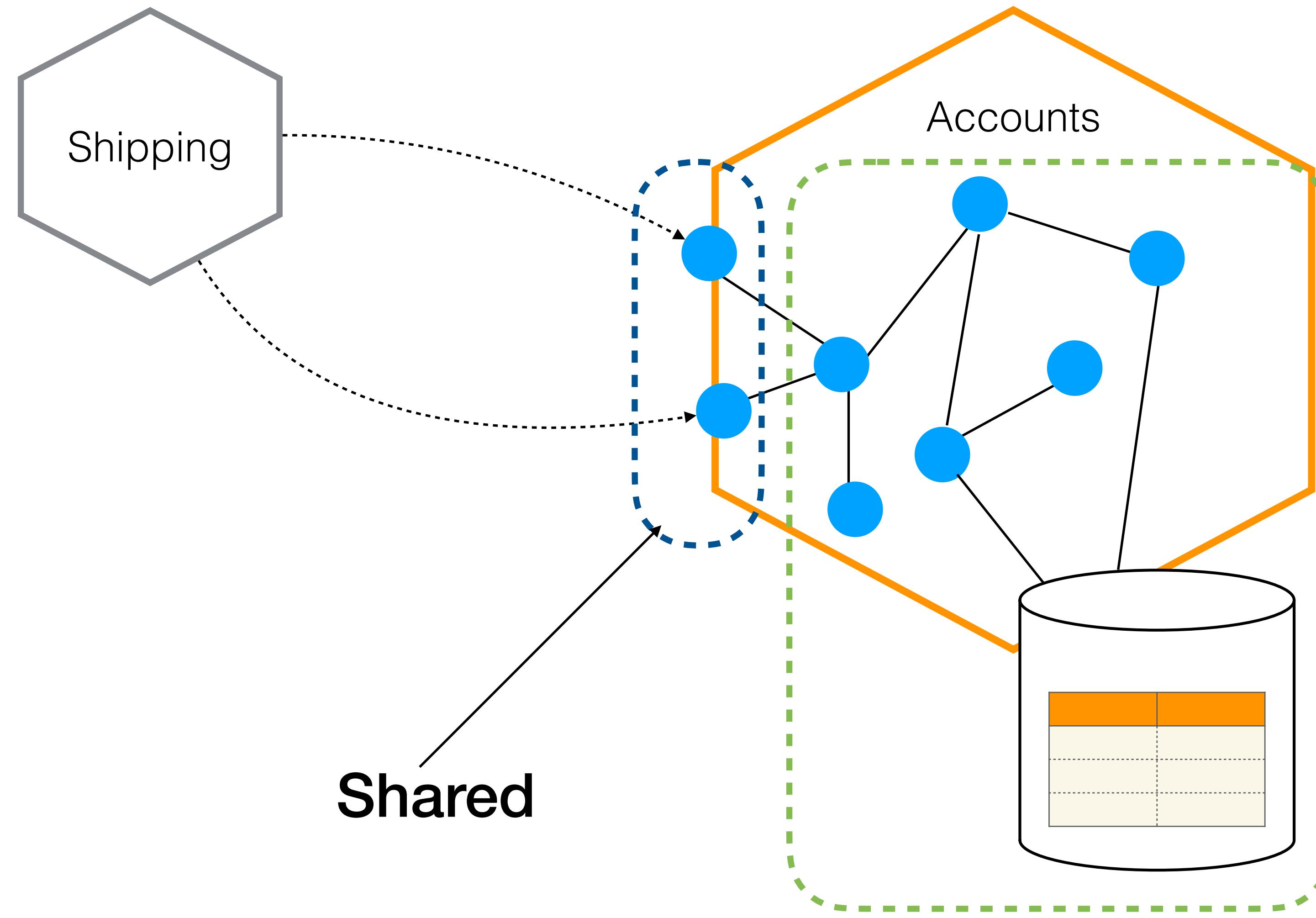
**HIDING!**



**Hide your secrets!**

**Be explicit about what is shared, and what is hidden**

**HIDING!**

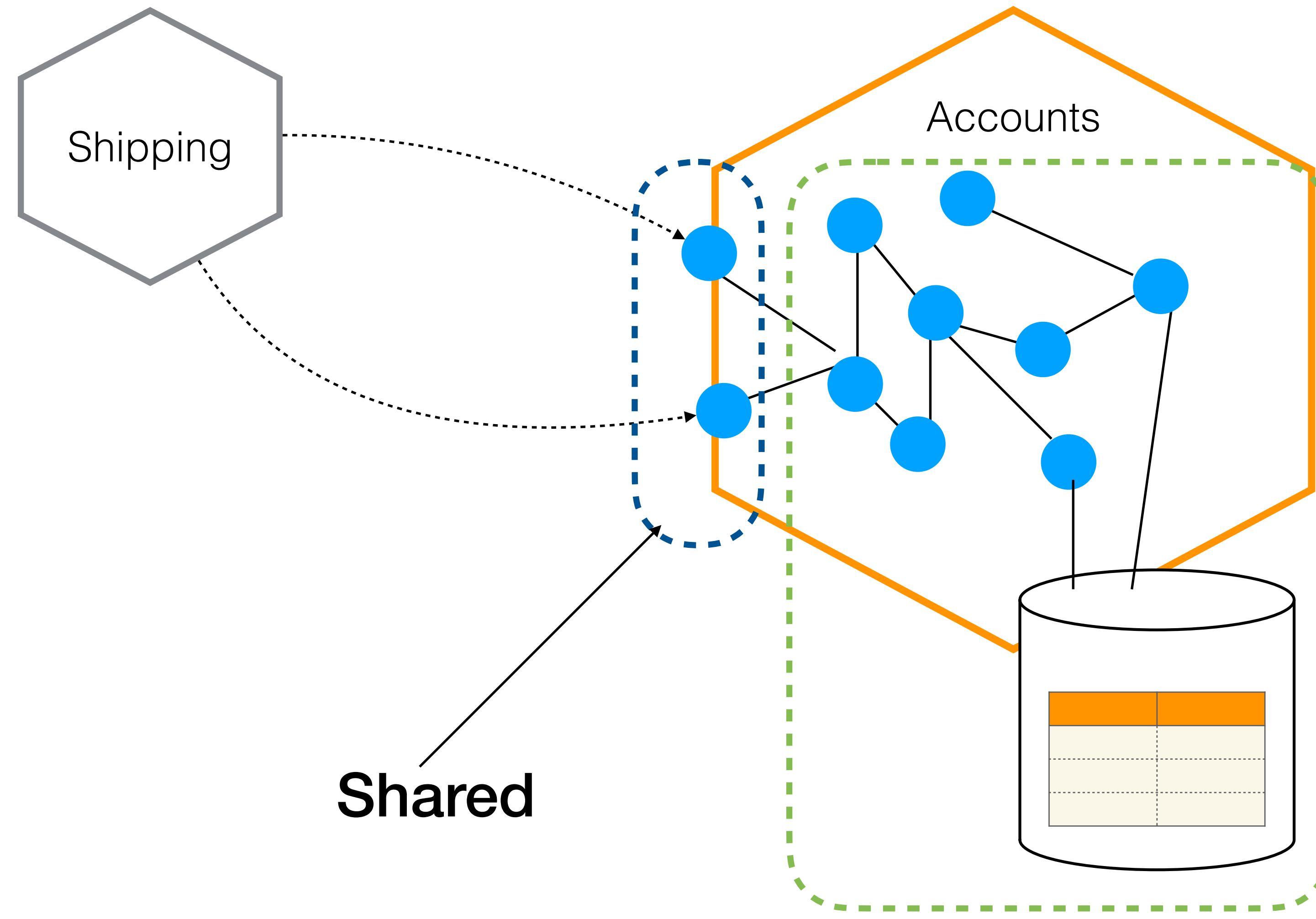


**Hide your secrets!**

**Be explicit about what is shared, and what is hidden**

**Hidden things can change, shared things can't**

# HIDING!



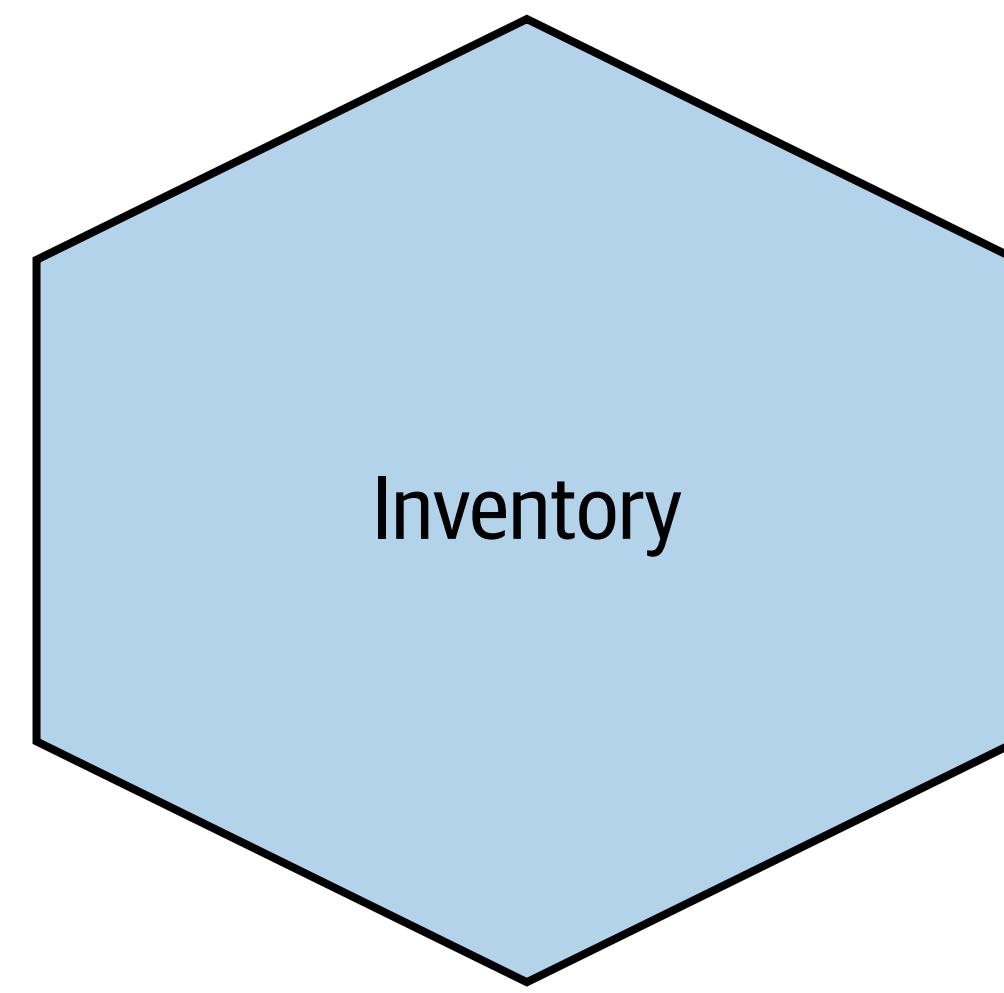
Hide your secrets!

Be explicit about what is shared, and what is hidden

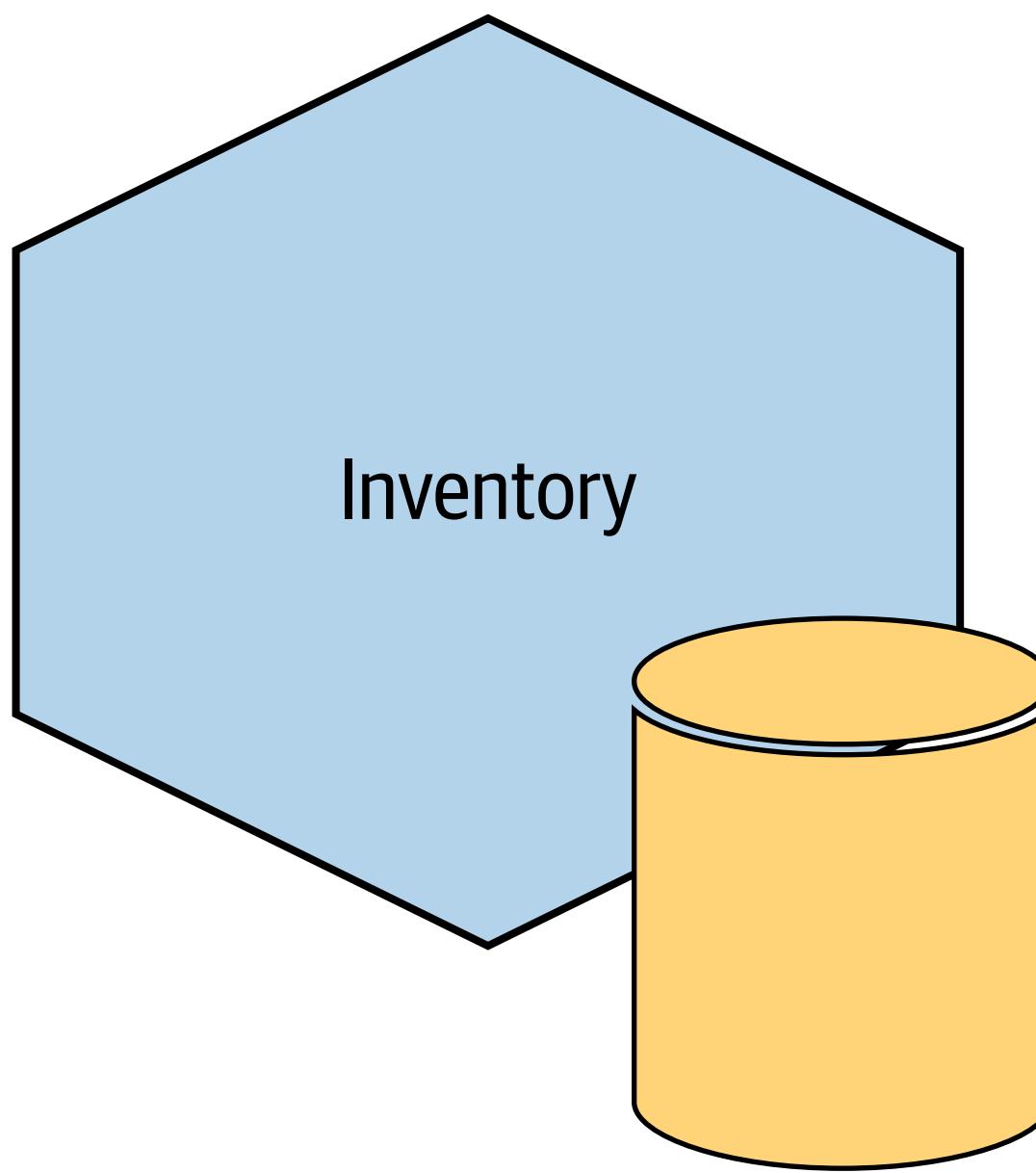
Hidden things can change, shared things can't

Hidden

## CONSUMER NEEDS...

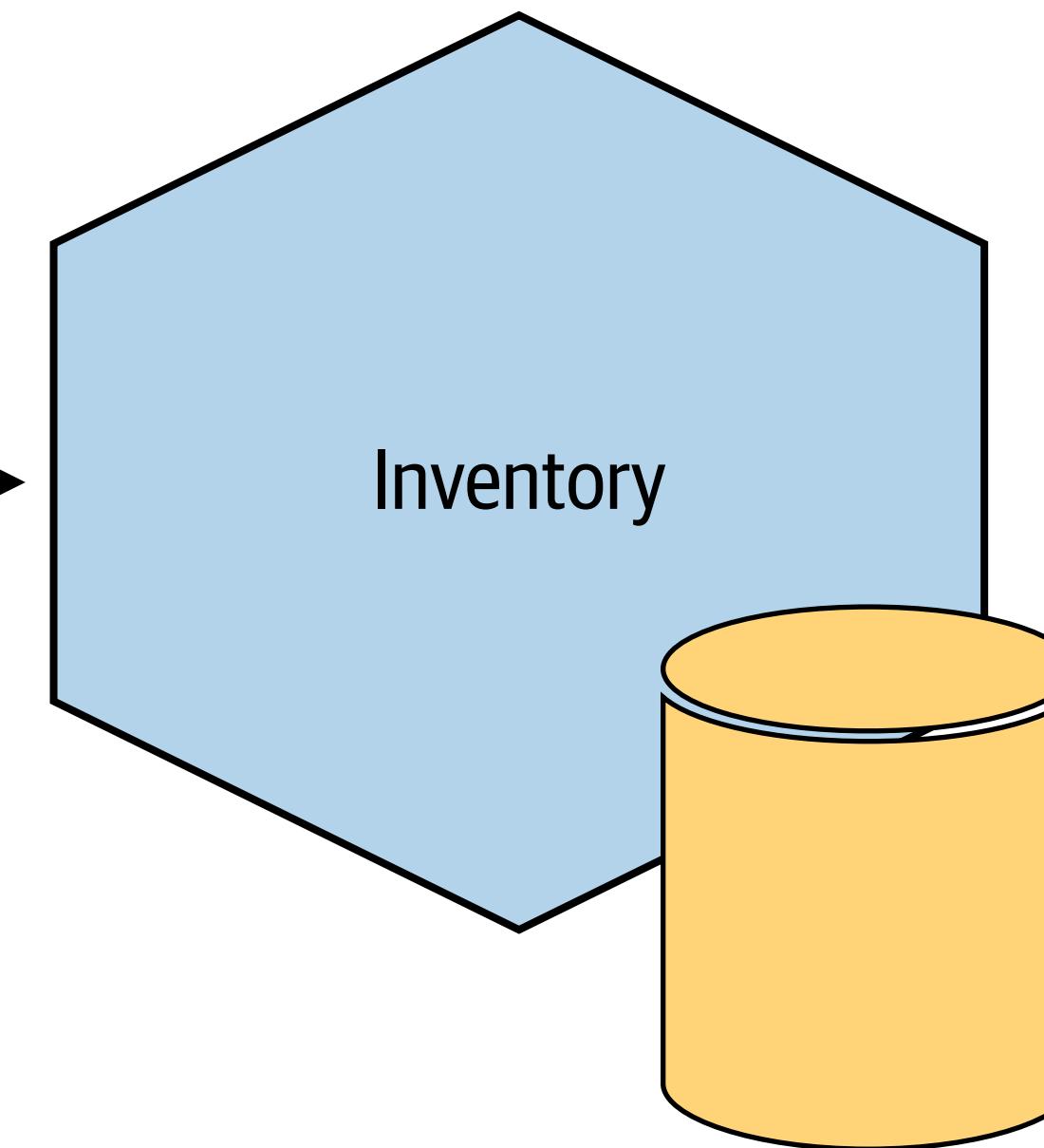


## CONSUMER NEEDS...



## CONSUMER NEEDS...

**How much Bieber do we have?**

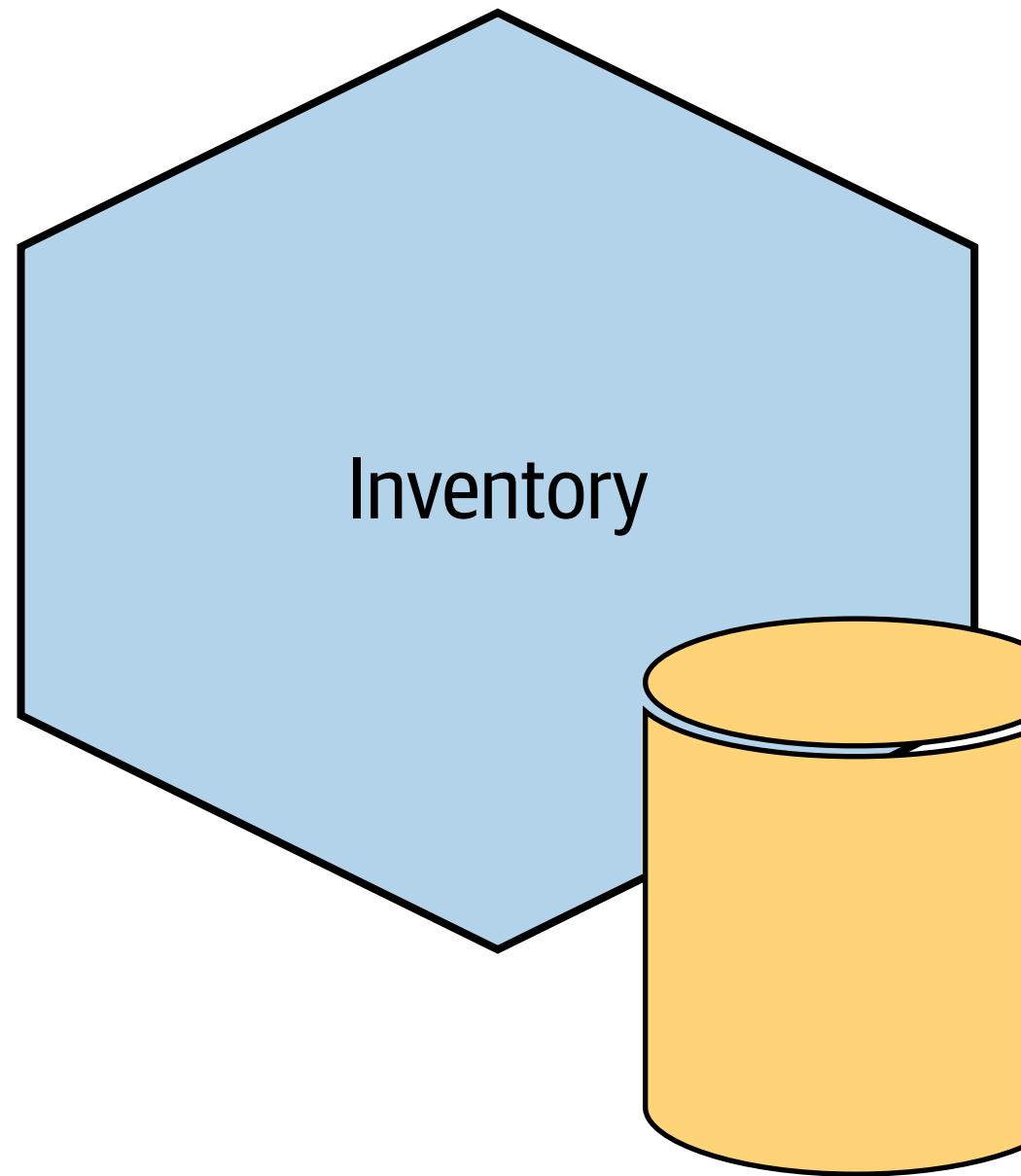


## CONSUMER NEEDS...

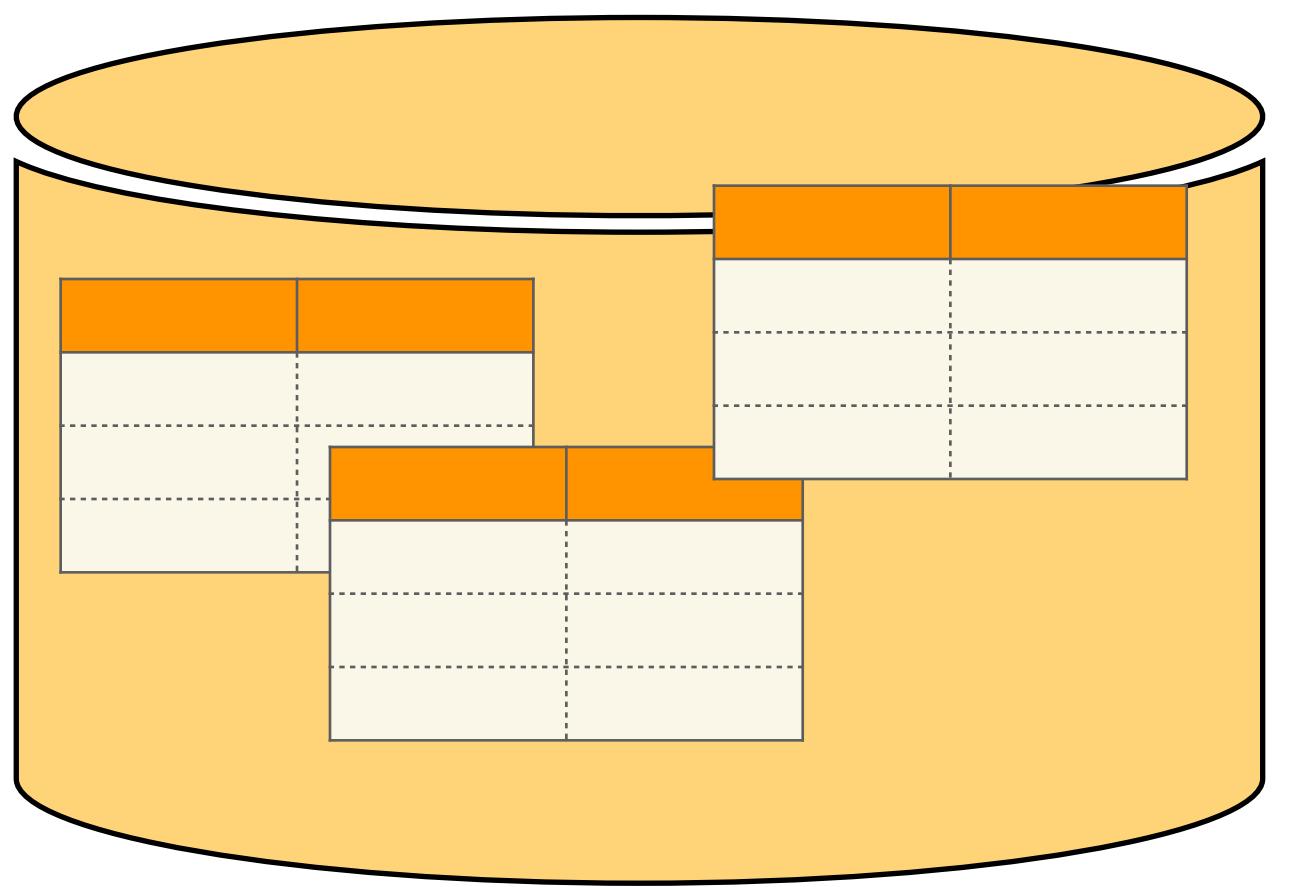
How much Bieber do we have?



```
{  
  sku: 123,  
  stock: 578,  
  name: Justin...  
  supplier: {  
    name: acme,  
    website: ...  
  }  
}
```

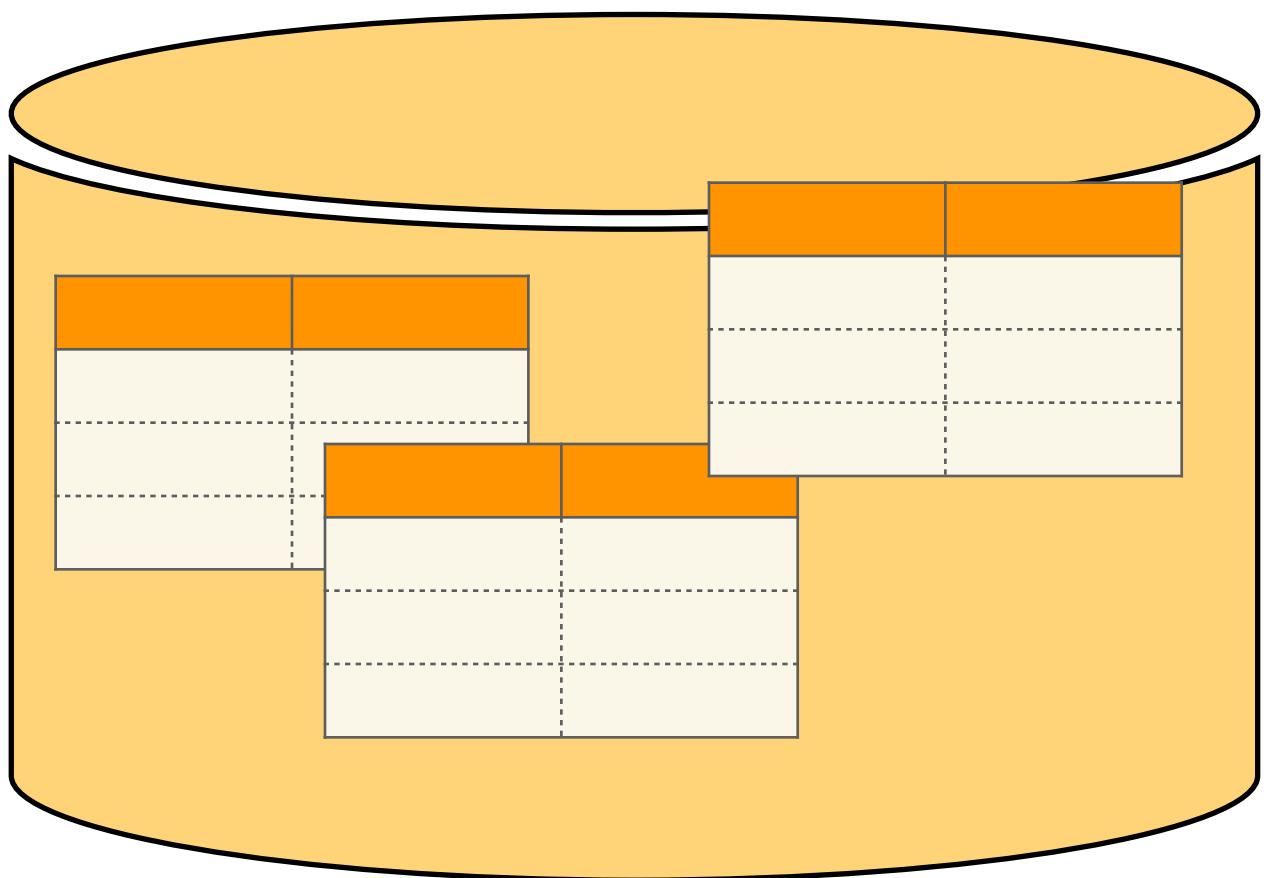


## EXAMPLE - INVENTORY MANAGEMENT



**Inventory**

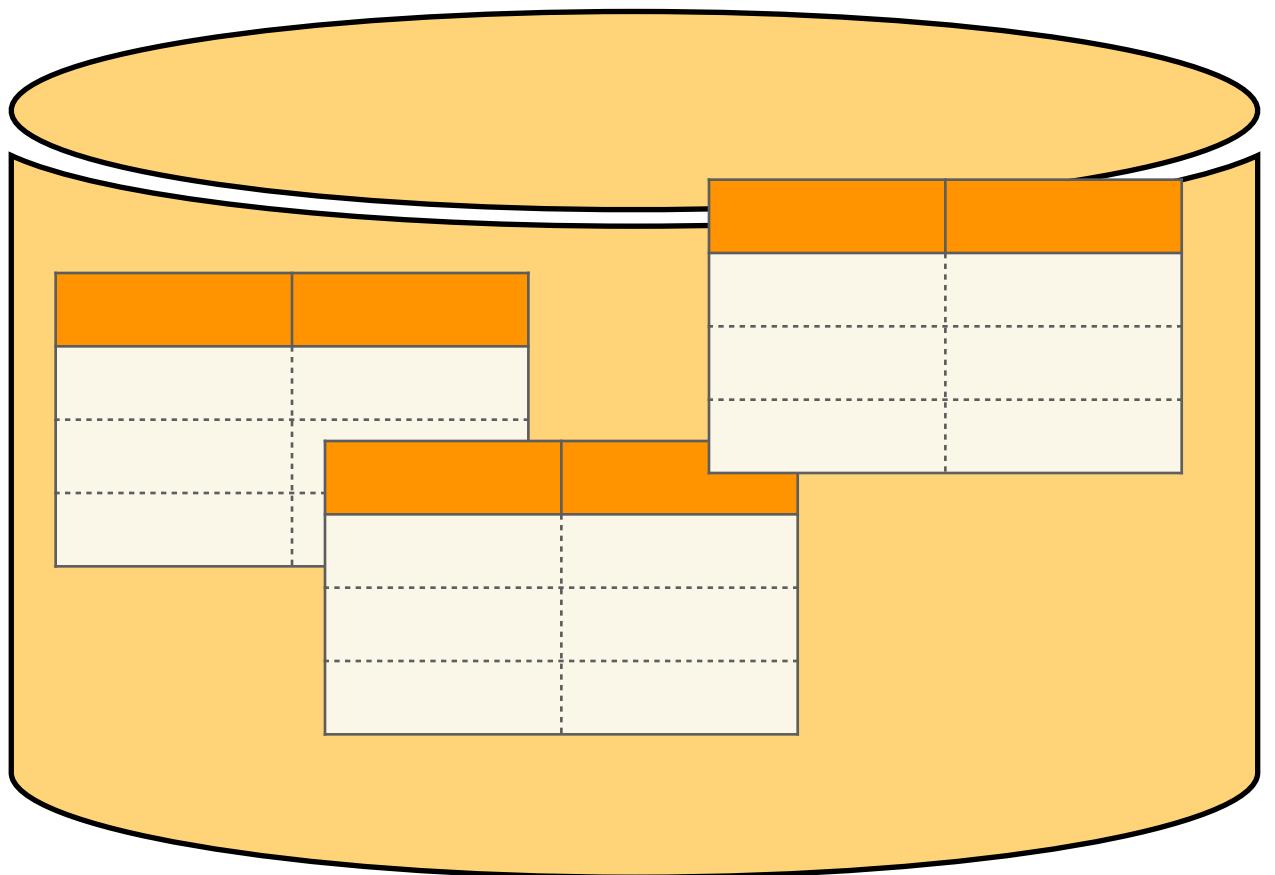
## EXAMPLE - INVENTORY MANAGEMENT



**Inventory**

SKU	Name	Supplier
123	Justin Bieber's Greatest Hits!	abc-1
<b>Item</b>		

## EXAMPLE - INVENTORY MANAGEMENT



**Inventory**

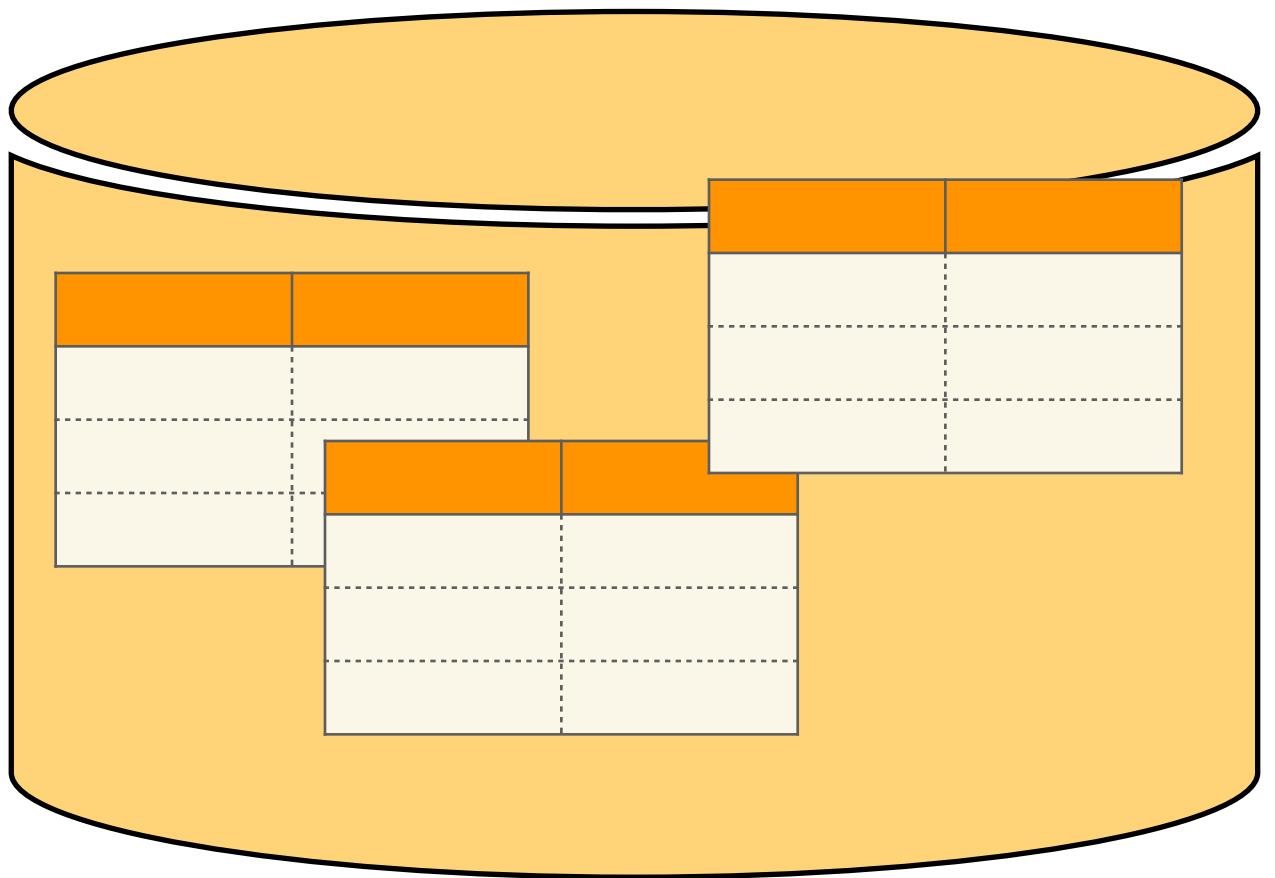
SKU	Row	Shelf	Number
123	17	6	128
123	100	12	450

**Location**

SKU	Name	Supplier
123	Justin Bieber's Greatest Hits!	abc-1

**Item**

## EXAMPLE - INVENTORY MANAGEMENT



**Inventory**

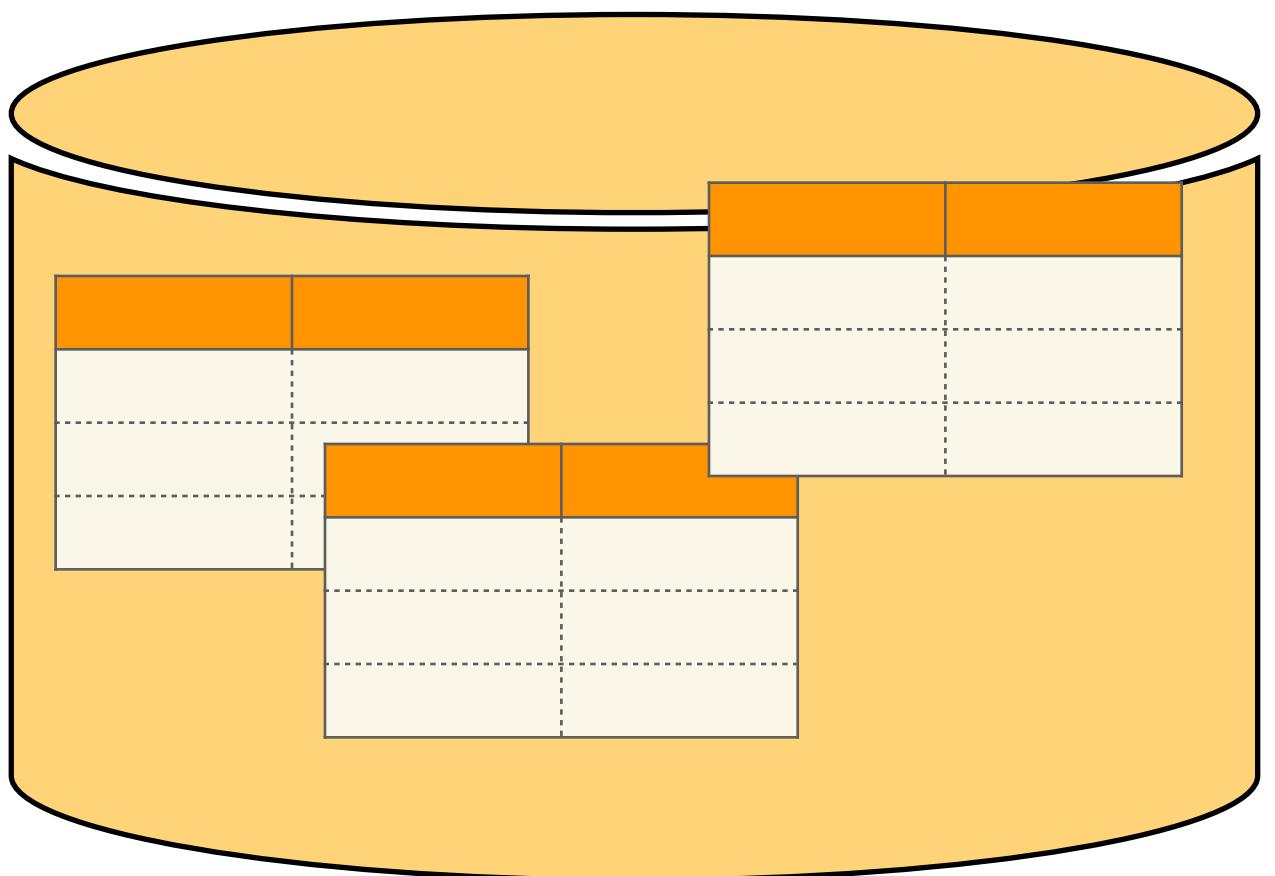
SKU	Row	Shelf	Number
123	17	6	128
123	100	12	450

**Location**

SKU	Name	Supplier
123	Justin Bieber's Greatest Hits!	abc-1

**Item**

## EXAMPLE - INVENTORY MANAGEMENT



**Inventory**

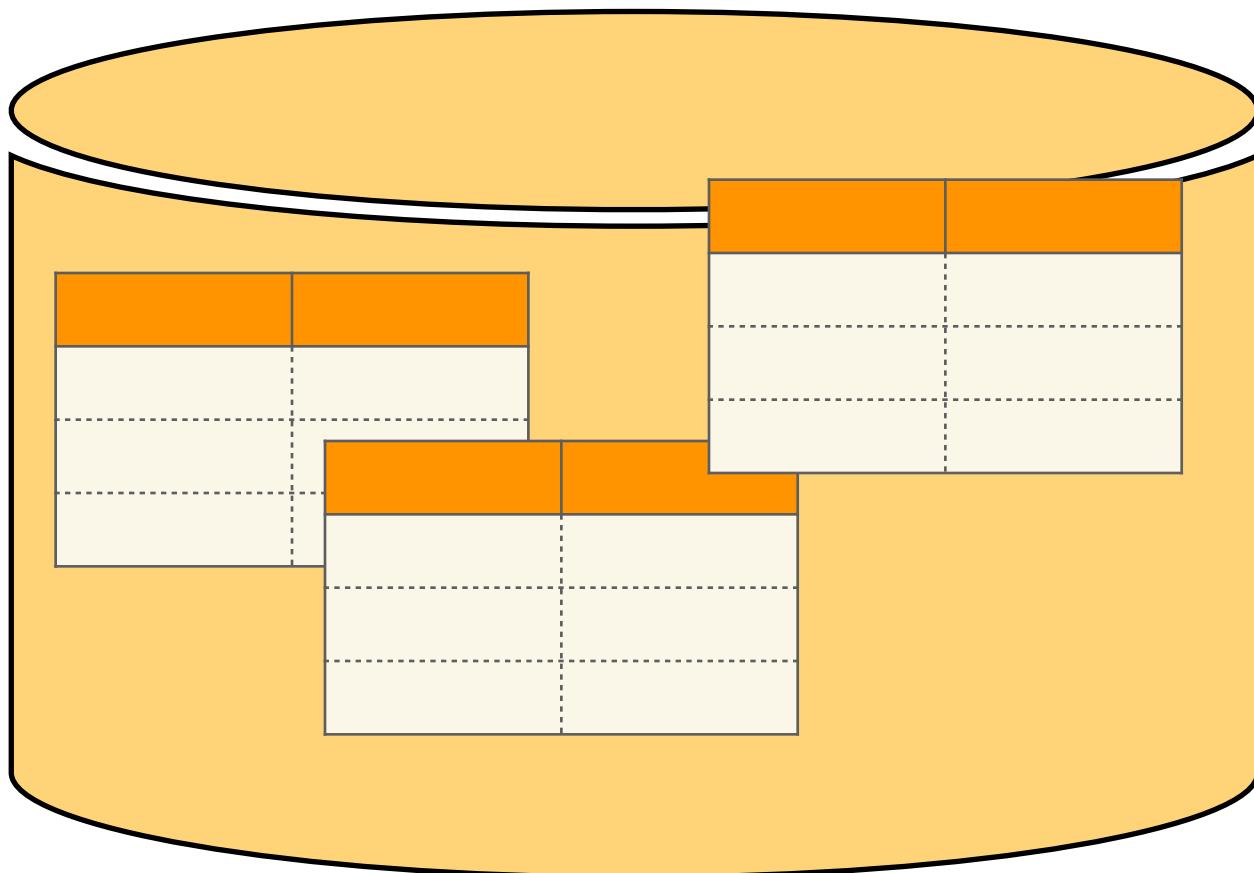
SKU	Row	Shelf	Number
123	17	6	128
123	100	12	450

**Location**

SKU	Name	Supplier
123	Justin Bieber's Greatest Hits!	abc-1

**Item**

# EXAMPLE - INVENTORY MANAGEMENT



**Inventory**

SKU	Row	Shelf	Number
123	17	6	128
123	100	12	450

**Location**

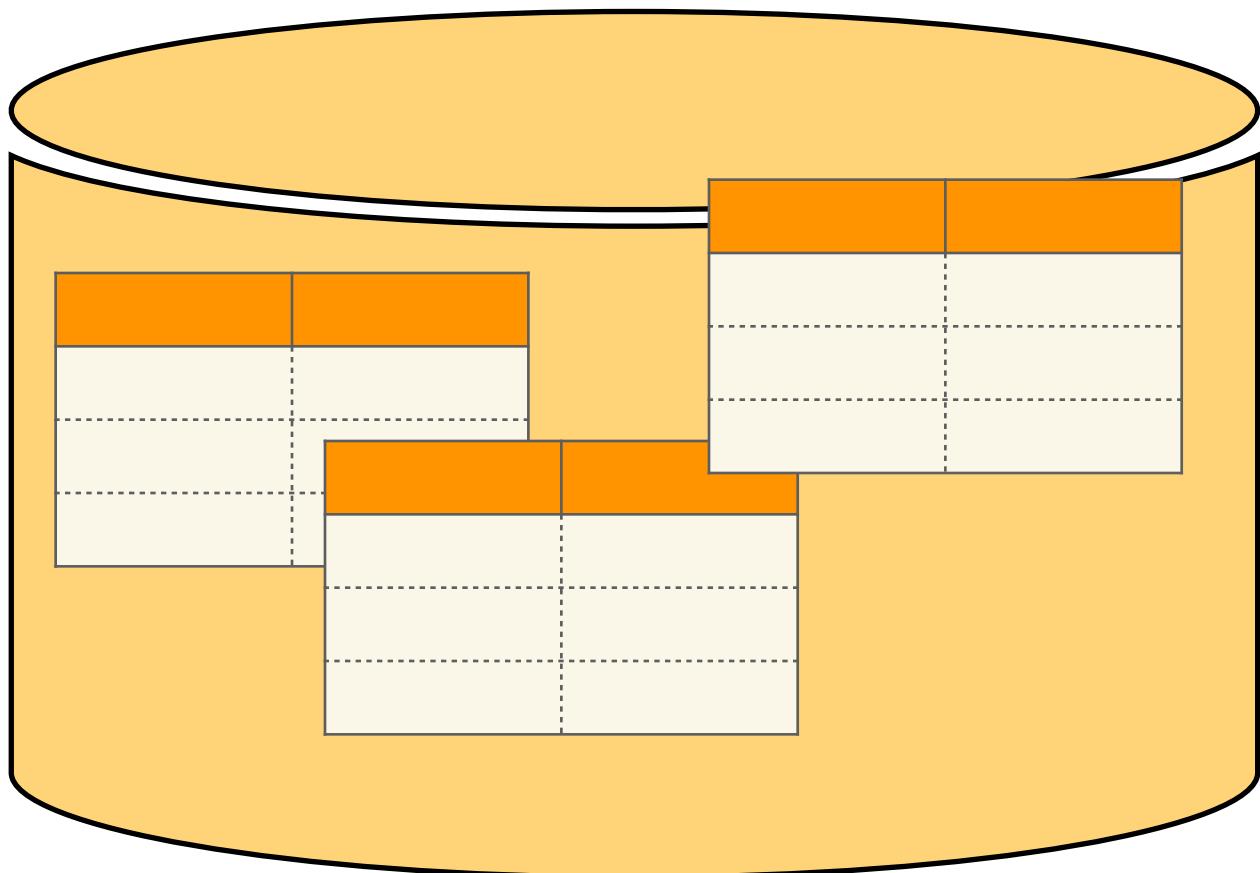
SKU	Name	Supplier
123	Justin Bieber's Greatest Hits!	abc-1

**Item**

ID	Company Name	Homepage	VAT	Contact
abc-1	Acme	...	...	...

**Supplier Table**

# EXAMPLE - INVENTORY MANAGEMENT



**Inventory**

SKU	Row	Shelf	Number
123	17	6	128
123	100	12	450

**Location**

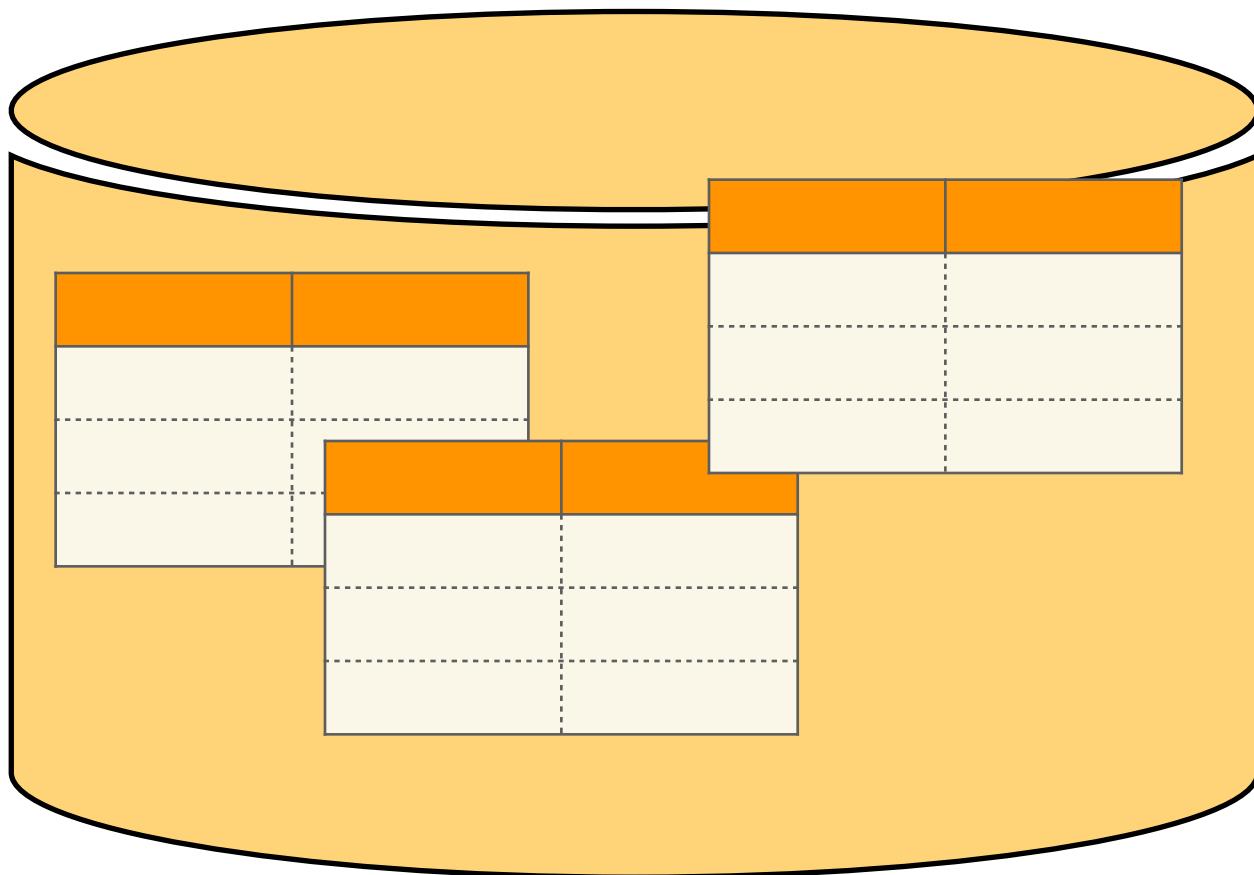
SKU	Name	Supplier
123	Justin Bieber's Greatest Hits!	abc-1

**Item**

ID	Company Name	Homepage	VAT	Contact
abc-1	Acme	...	...	...

**Supplier Table**

# EXAMPLE - INVENTORY MANAGEMENT



**Inventory**

SKU	Row	Shelf	Number
123	17	6	128
123	100	12	450

**Location**

SKU	Name	Supplier
123	Justin Bieber's Greatest Hits!	abc-1

**Item**

ID	Company Name	Homepage	VAT	Contact
abc-1	Acme	...	...	...

**Supplier Table**

## LOTS OF DETAIL CAN BE HIDDEN...

SKU	Row	Shelf	Number
123	17	6	128
123	100	12	450

**Location**

SKU	Name	Supplier
123	Justin Bieber's Greatest Hits!	abc-1

**Item**

ID	Company Name	Homepage	VAT	Contact
abc-1	Acme	...	...	...

**Supplier Table**

## LOTS OF DETAIL CAN BE HIDDEN...

```
{  
  sku: 123,  
  stock: 578,  
  name: Justin...  
  supplier: {  
    name: acme,  
    website: ...  
  }  
}
```

SKU	Row	Shelf	Number
123	17	6	128
123	100	12	450

**Location**

SKU	Name	Supplier
123	Justin Bieber's Greatest Hits!	abc-1

**Item**

ID	Company Name	Homepage	VAT	Contact
abc-1	Acme	...	...	...

**Supplier Table**

## LOTS OF DETAIL CAN BE HIDDEN...

```
{  
  sku: 123,  
  stock: 578,  
  name: Justin...  
  supplier: {  
    name: acme,  
    website: ...  
  }  
}
```

SKU	Row	Shelf	Number
123	17	6	128
123	100	12	450

Location

SKU	Name	Supplier
123	Justin Bieber's Greatest Hits!	abc-1

Item

ID	Company Name	Homepage	VAT	Contact
abc-1	Acme	...	...	...

Supplier Table

## LOTS OF DETAIL CAN BE HIDDEN...

```
{  
  sku: 123,  
  stock: 578,  
  name: Justin...  
  supplier: {  
    name: acme,  
    website: ...  
  }  
}
```

SKU	Row	Shelf	Number
123	17	6	128
123	100	12	450

Location

SKU	Name	Supplier
123	Justin Bieber's Greatest Hits!	abc-1

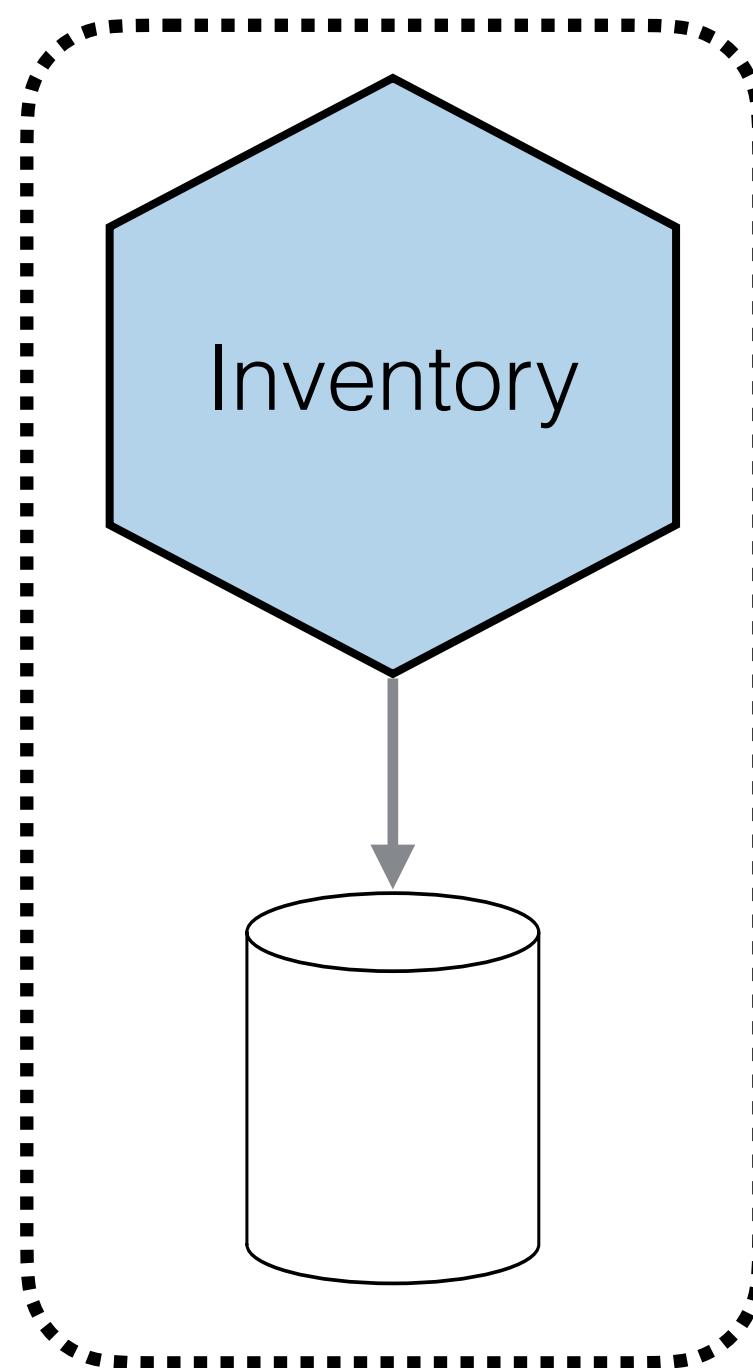
Item

ID	Company Name	Homepage	VAT	Contact
abc-1	Acme	...	...	...

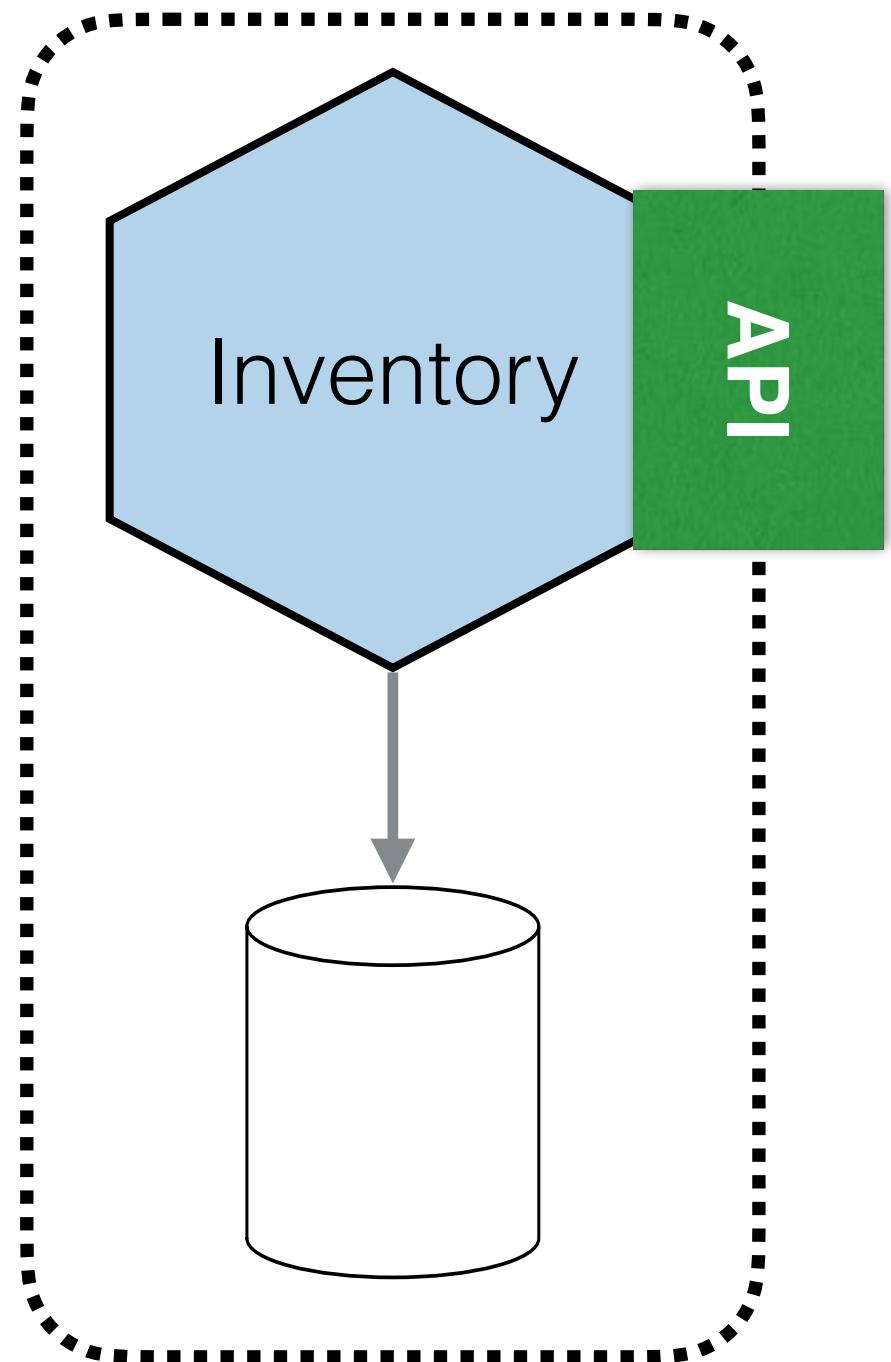
Supplier Table

# **What if I need direct DB access?**

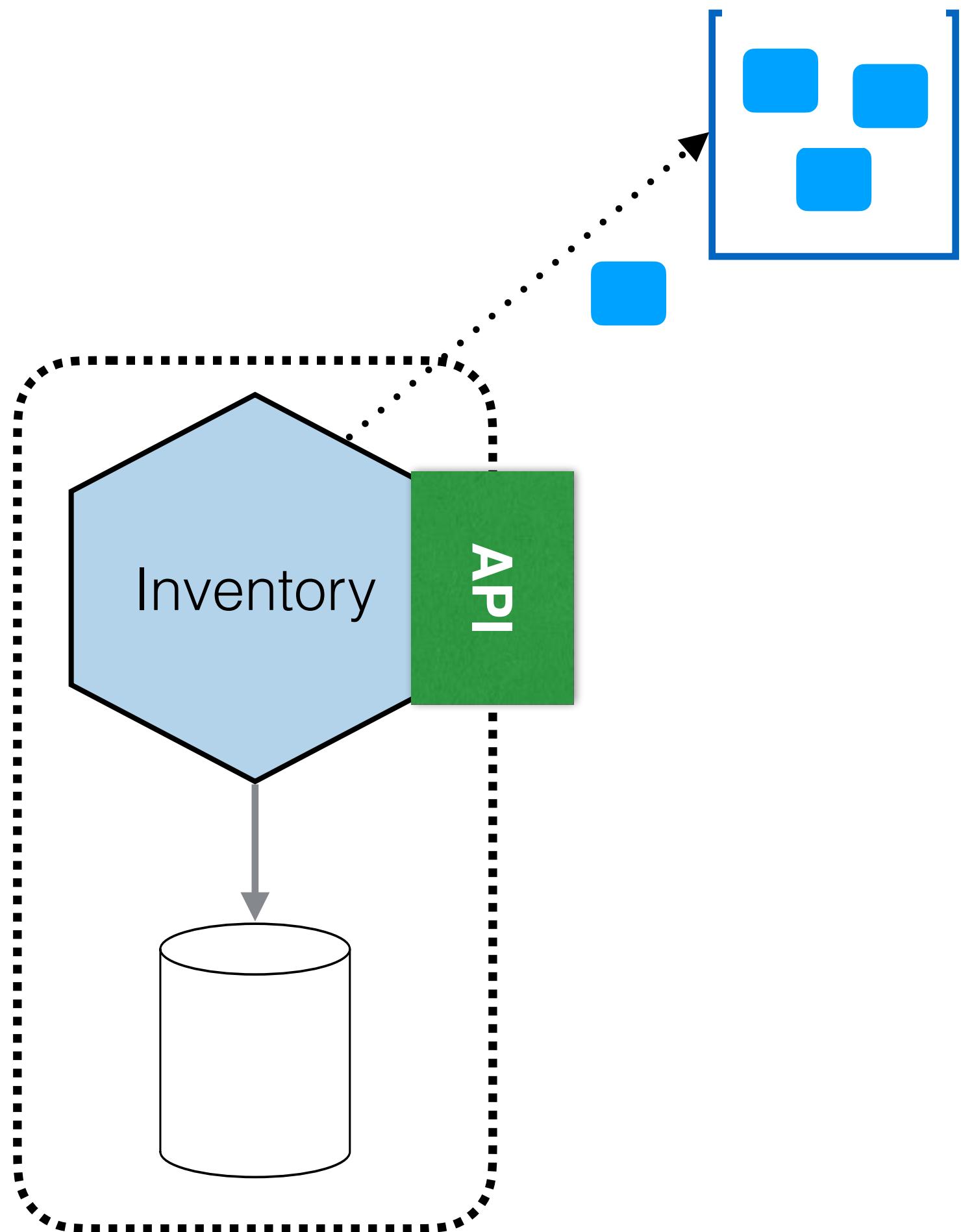
## PATTERN: DATABASE AS AN ENDPOINT



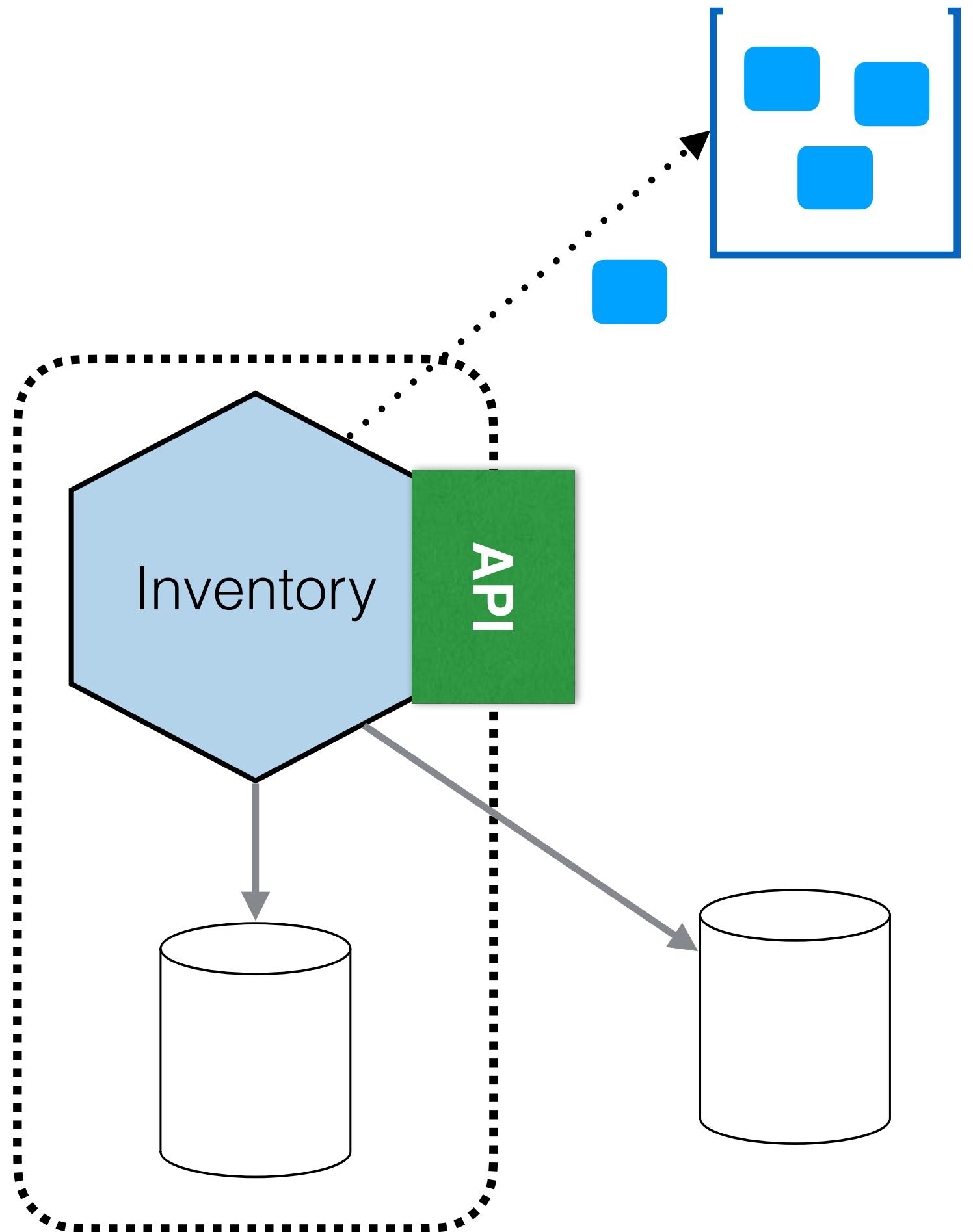
## PATTERN: DATABASE AS AN ENDPOINT



## PATTERN: DATABASE AS AN ENDPOINT

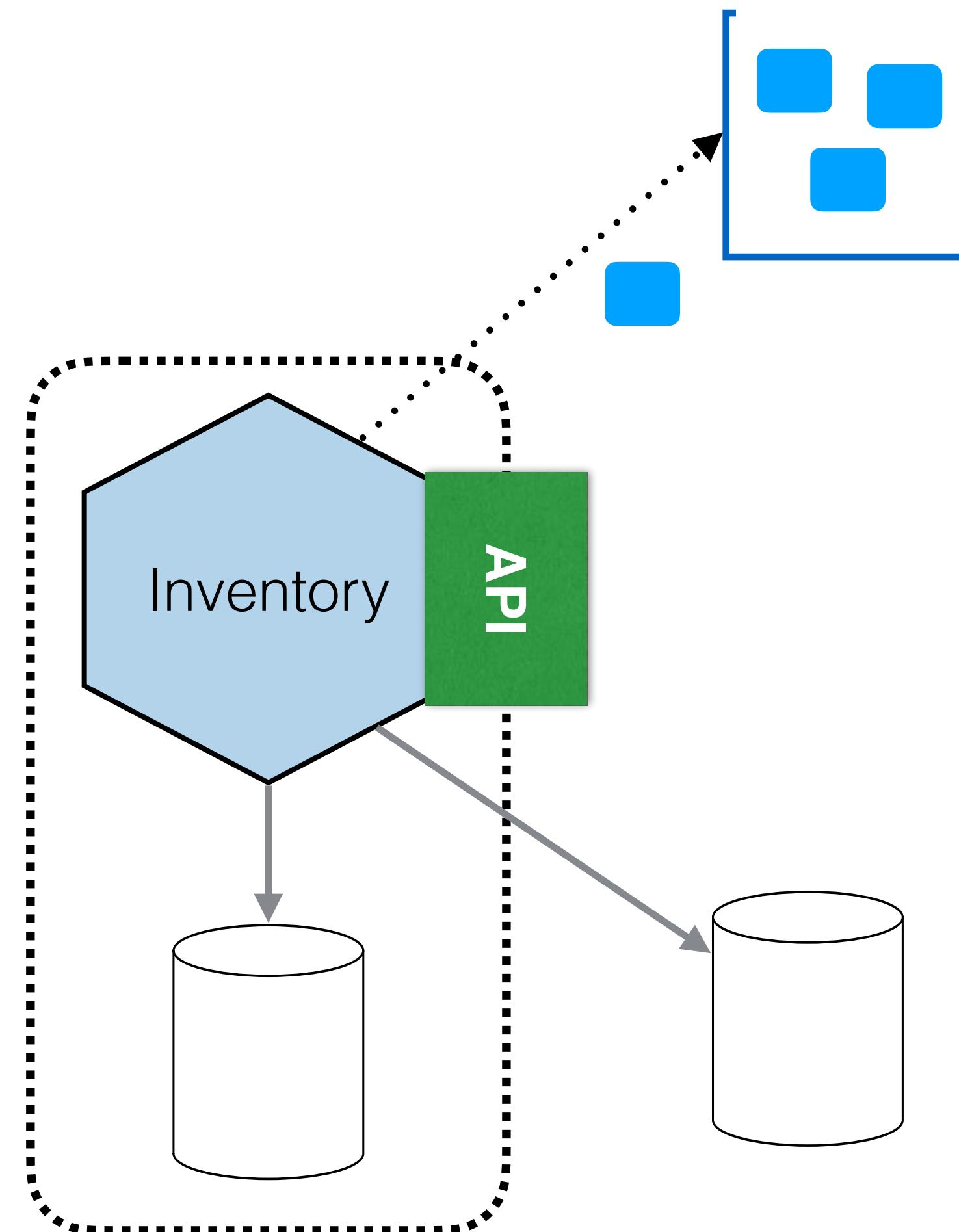


## PATTERN: DATABASE AS AN ENDPOINT



Expose a full DB as an endpoint

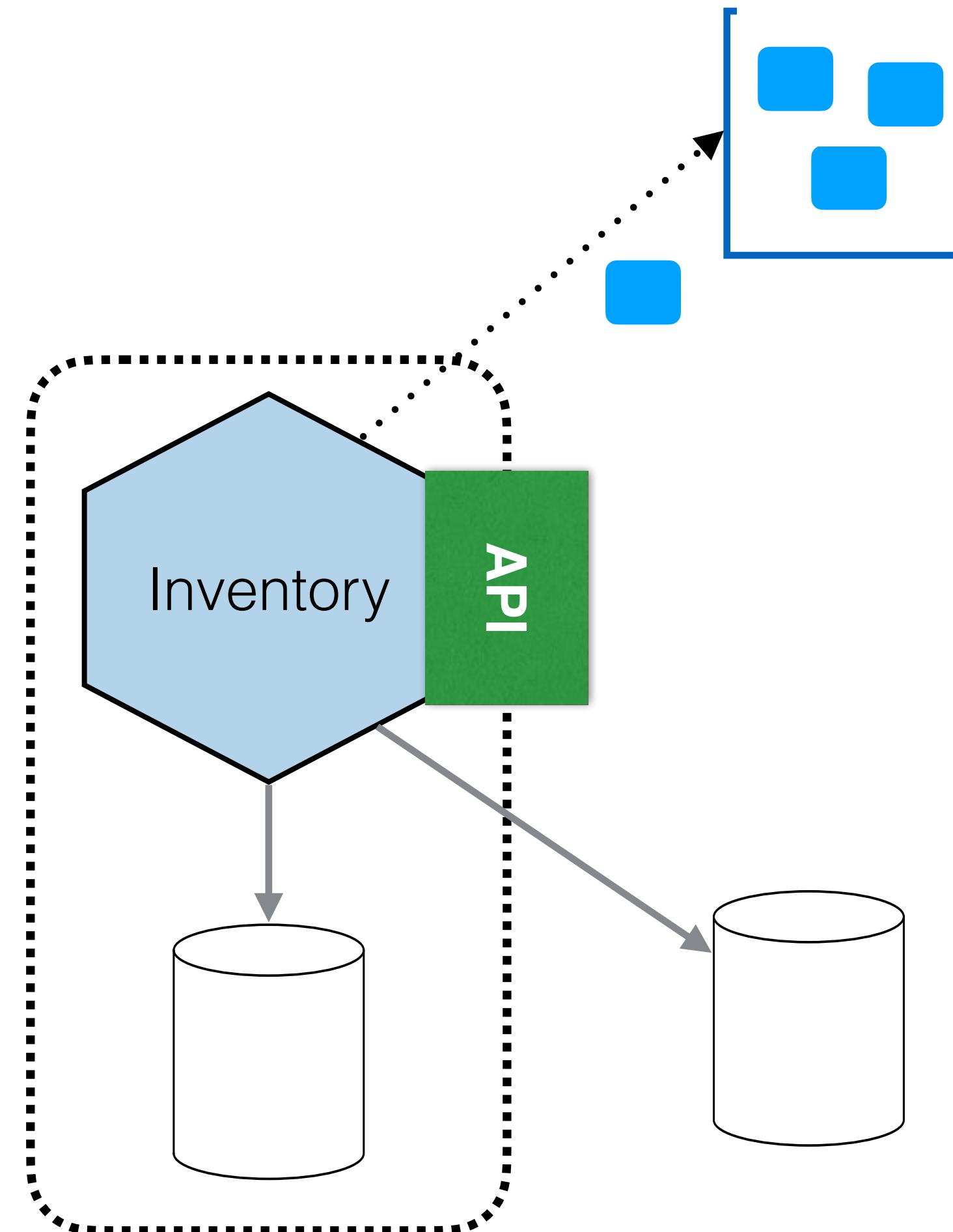
## PATTERN: DATABASE AS AN ENDPOINT



Expose a full DB as an endpoint

Views are one way to implement this pattern

## PATTERN: DATABASE AS AN ENDPOINT

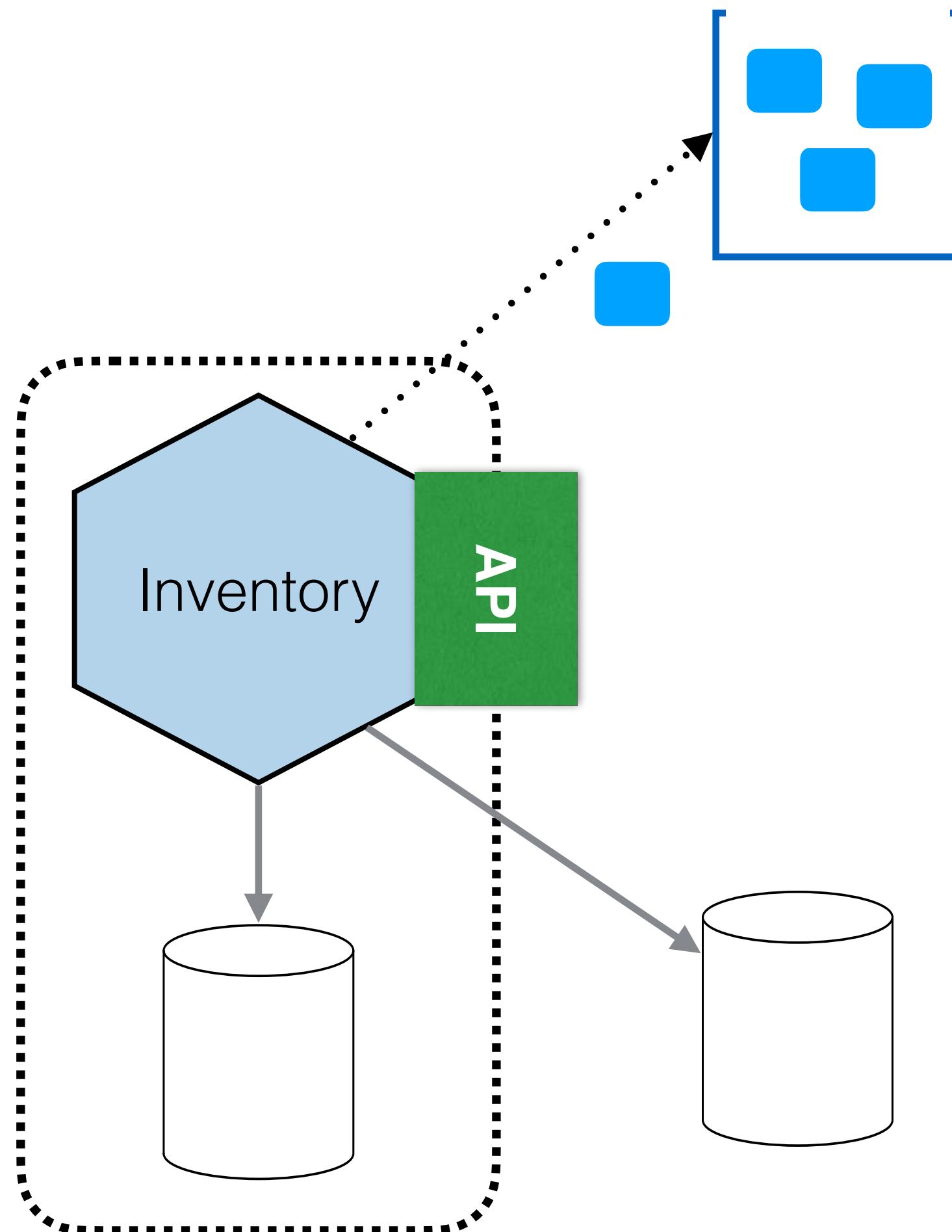


Expose a full DB as an endpoint

Views are one way to implement this pattern

Read-only

## PATTERN: DATABASE AS AN ENDPOINT



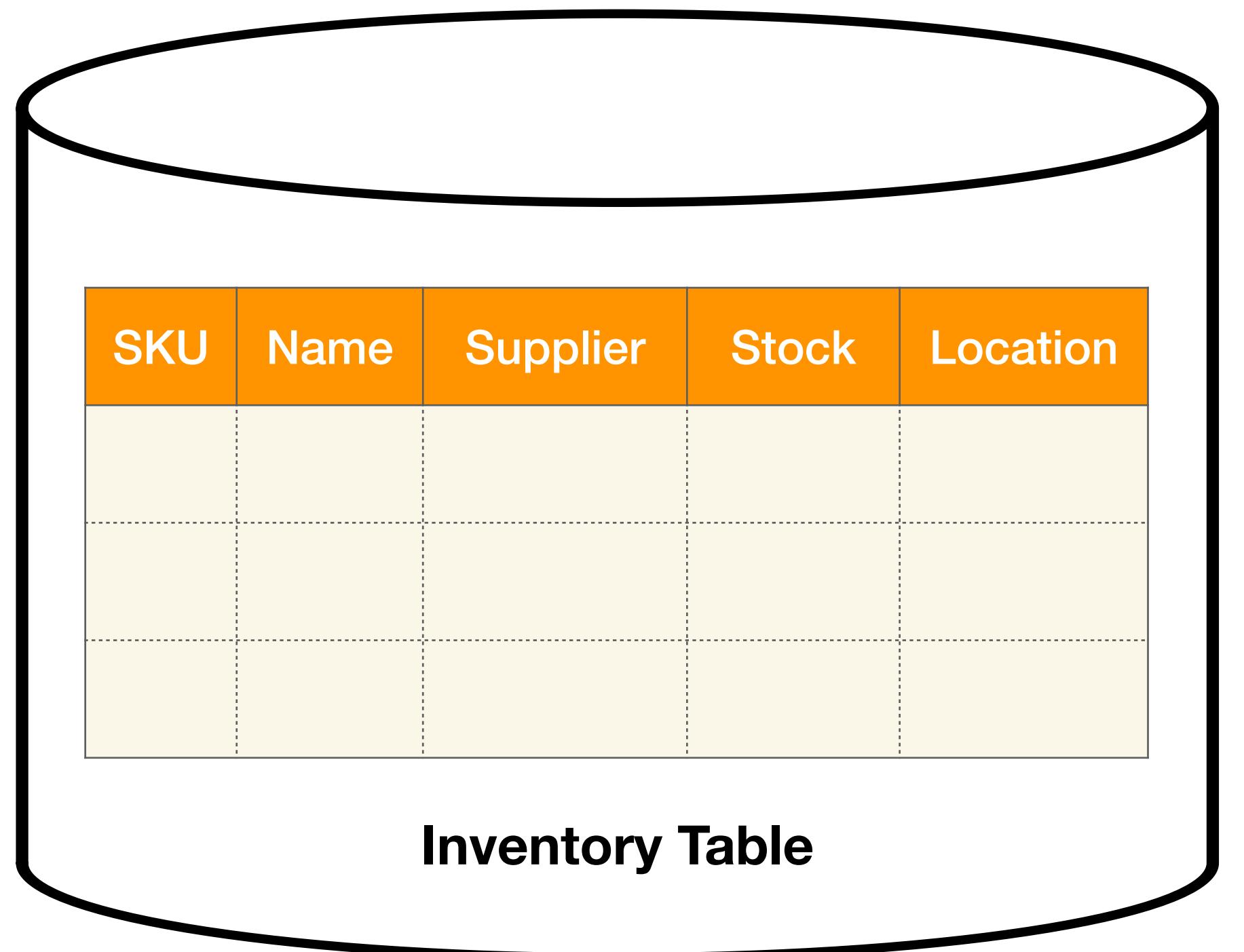
**Expose a full DB as an endpoint**

**Views are one way to implement this pattern**

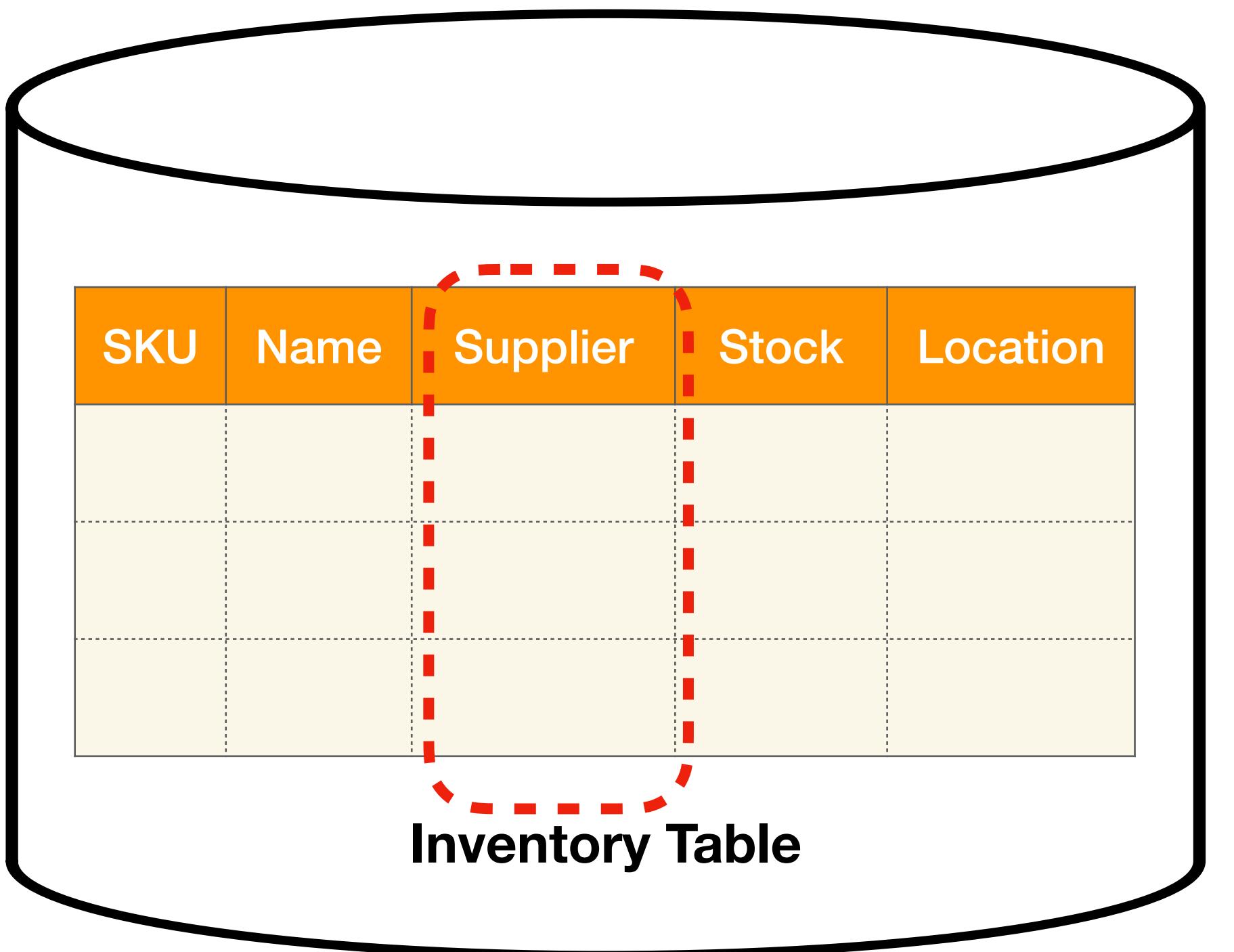
**Read-only**

**Can allow for expensive, ad-hoc queries more easily (e.g. joins)**

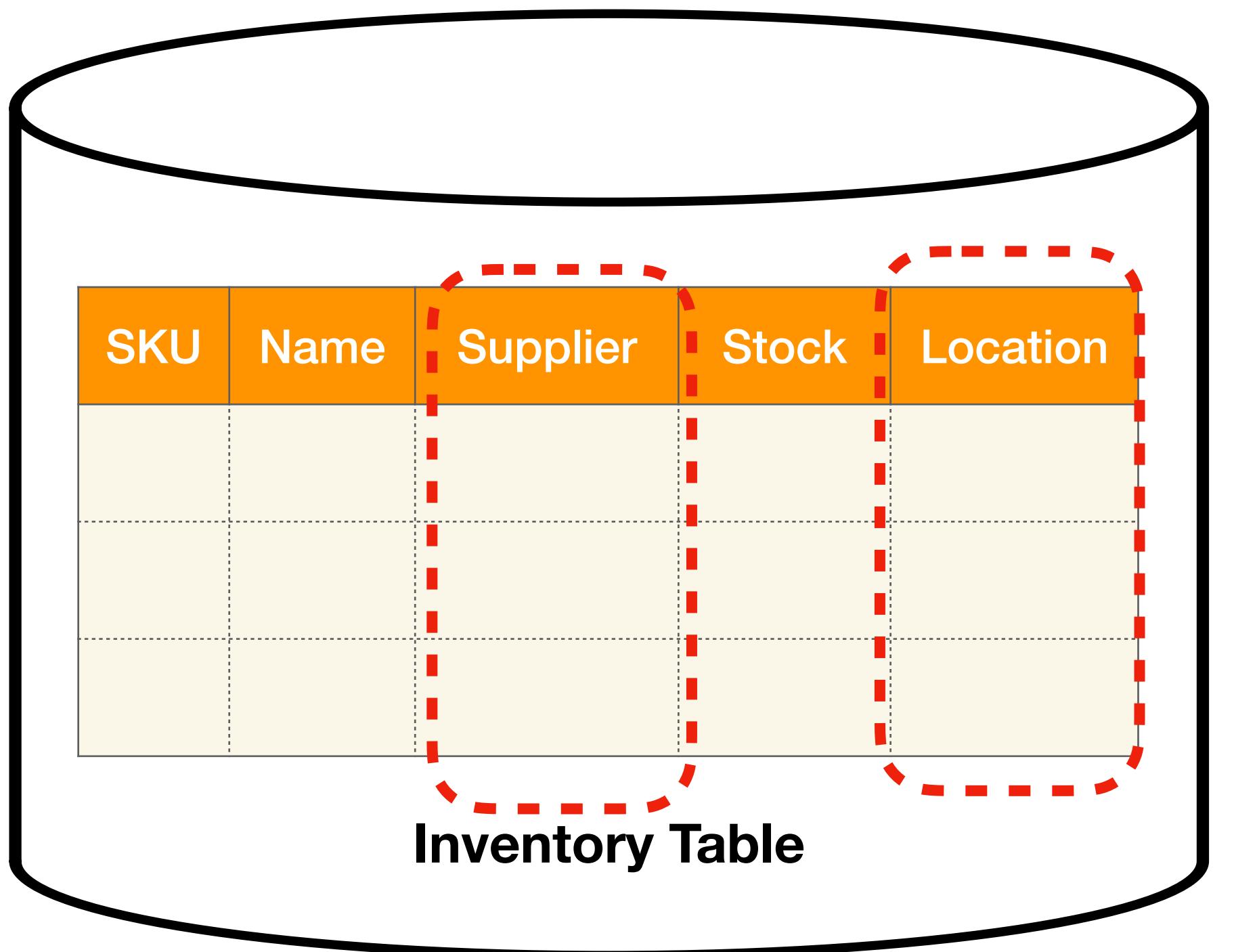
## PATTERN: DATABASE VIEWS



## PATTERN: DATABASE VIEWS

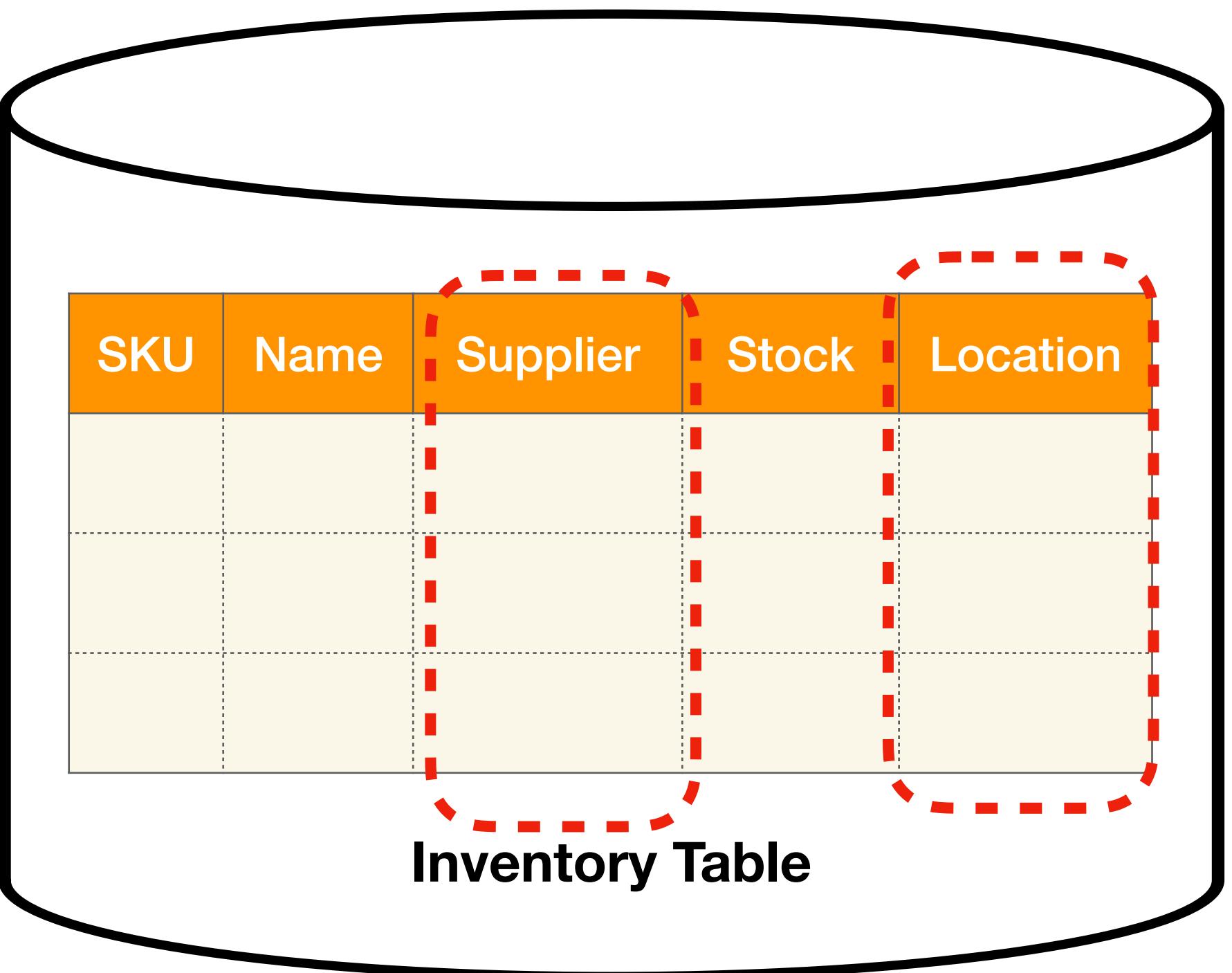


## PATTERN: DATABASE VIEWS



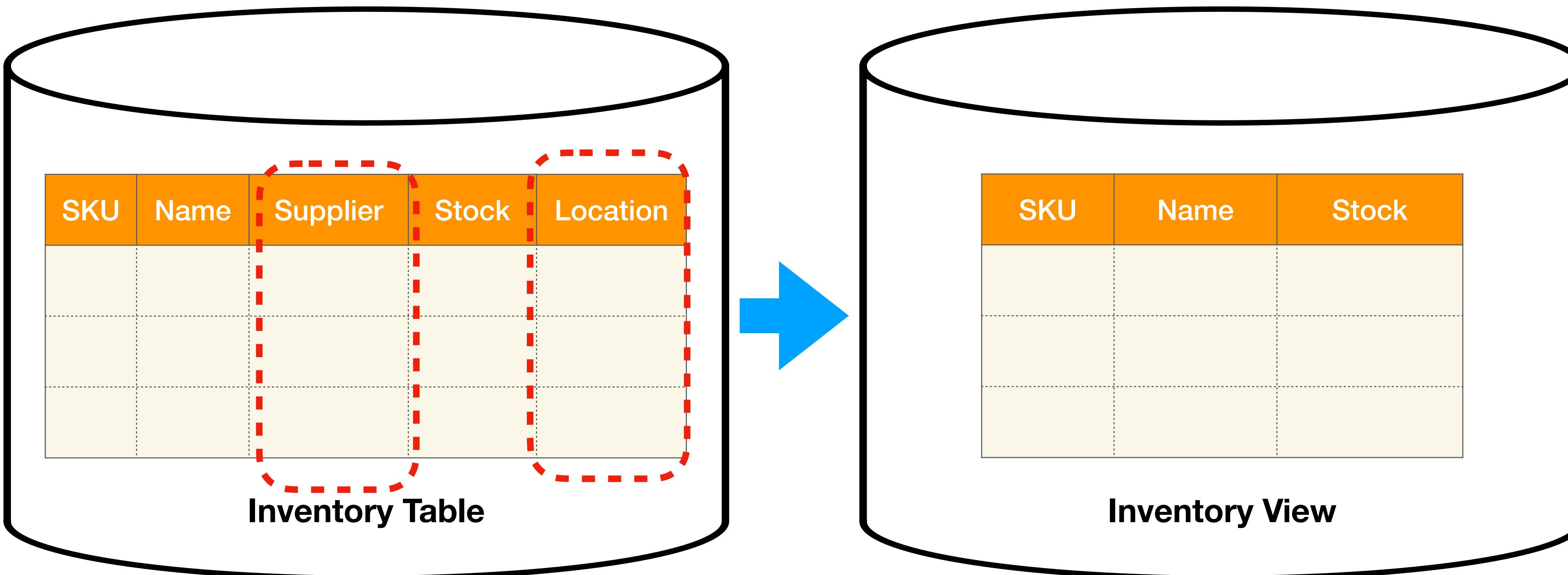
## PATTERN: DATABASE VIEWS

Project a subset of a database to a read-only view



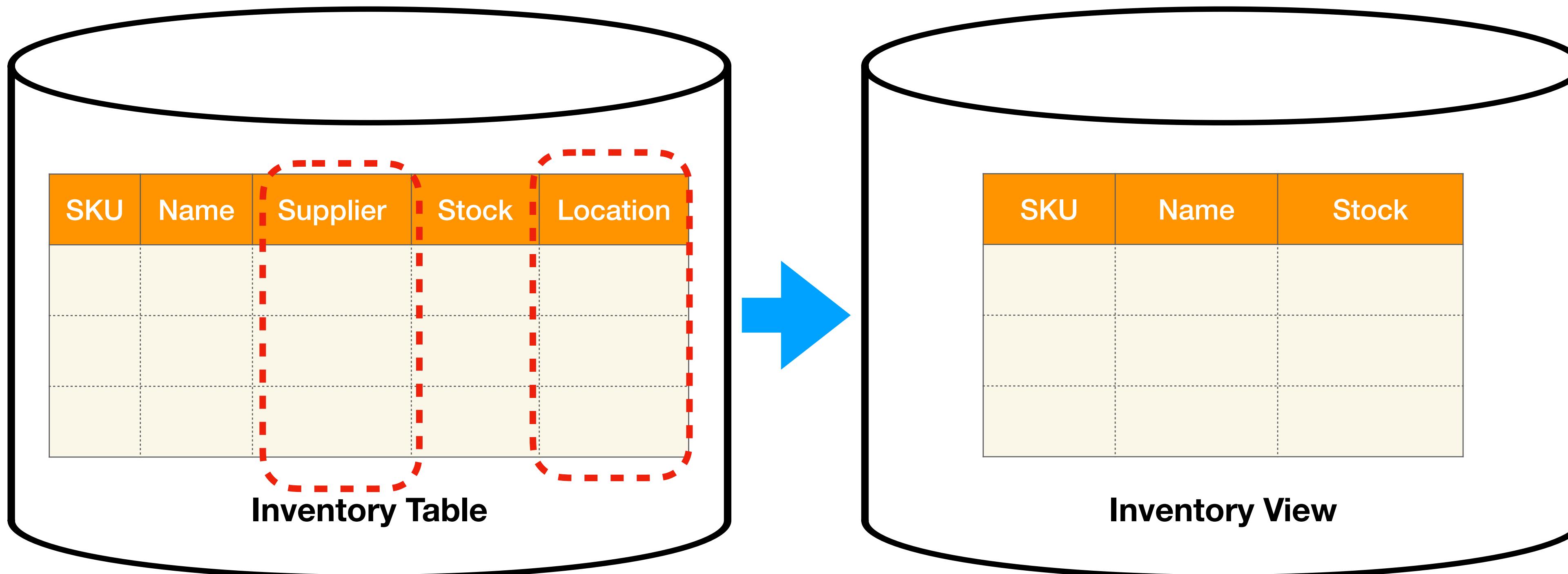
## PATTERN: DATABASE VIEWS

Project a subset of a database to a read-only view



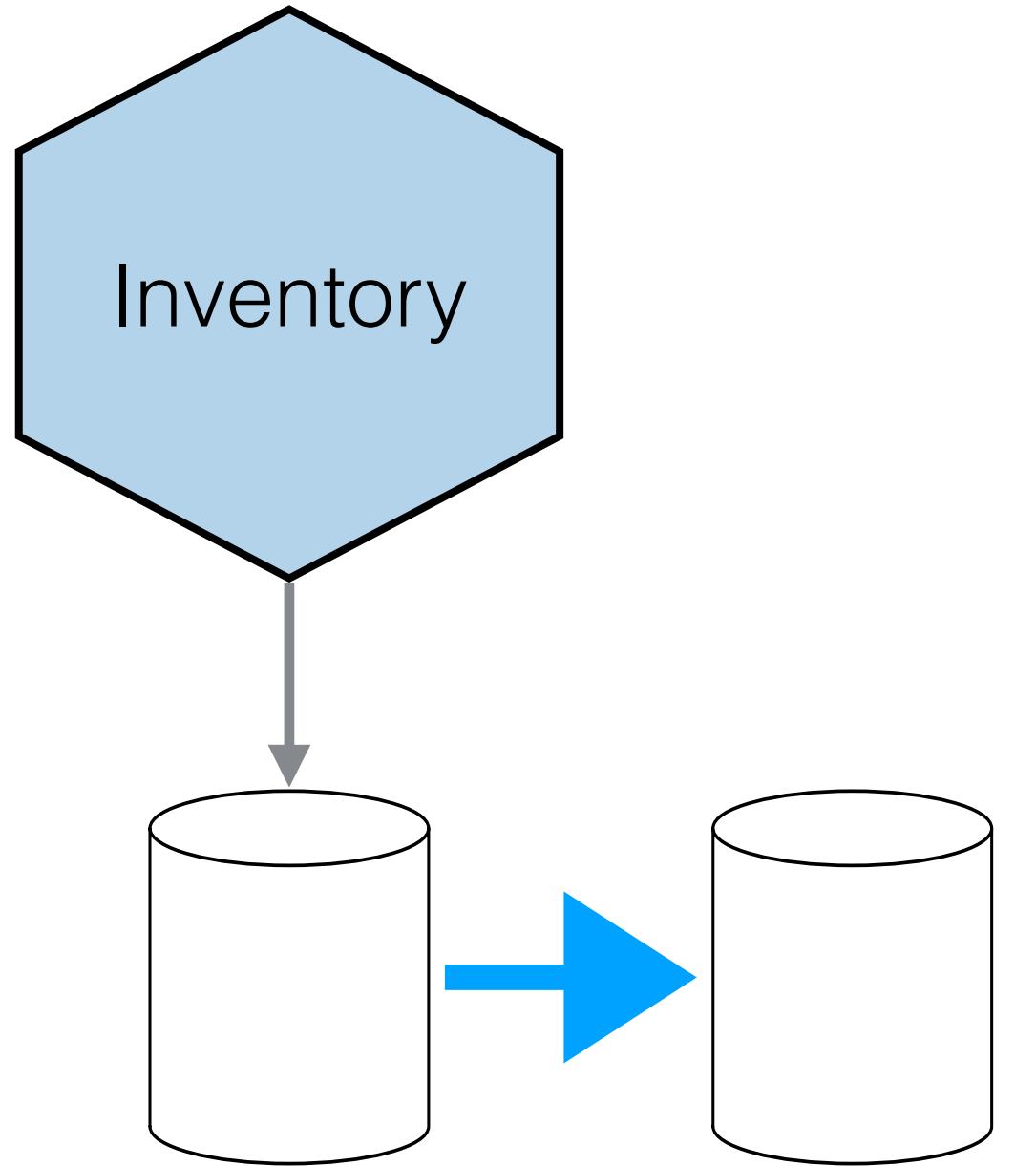
## PATTERN: DATABASE VIEWS

Project a subset of a database to a read-only view

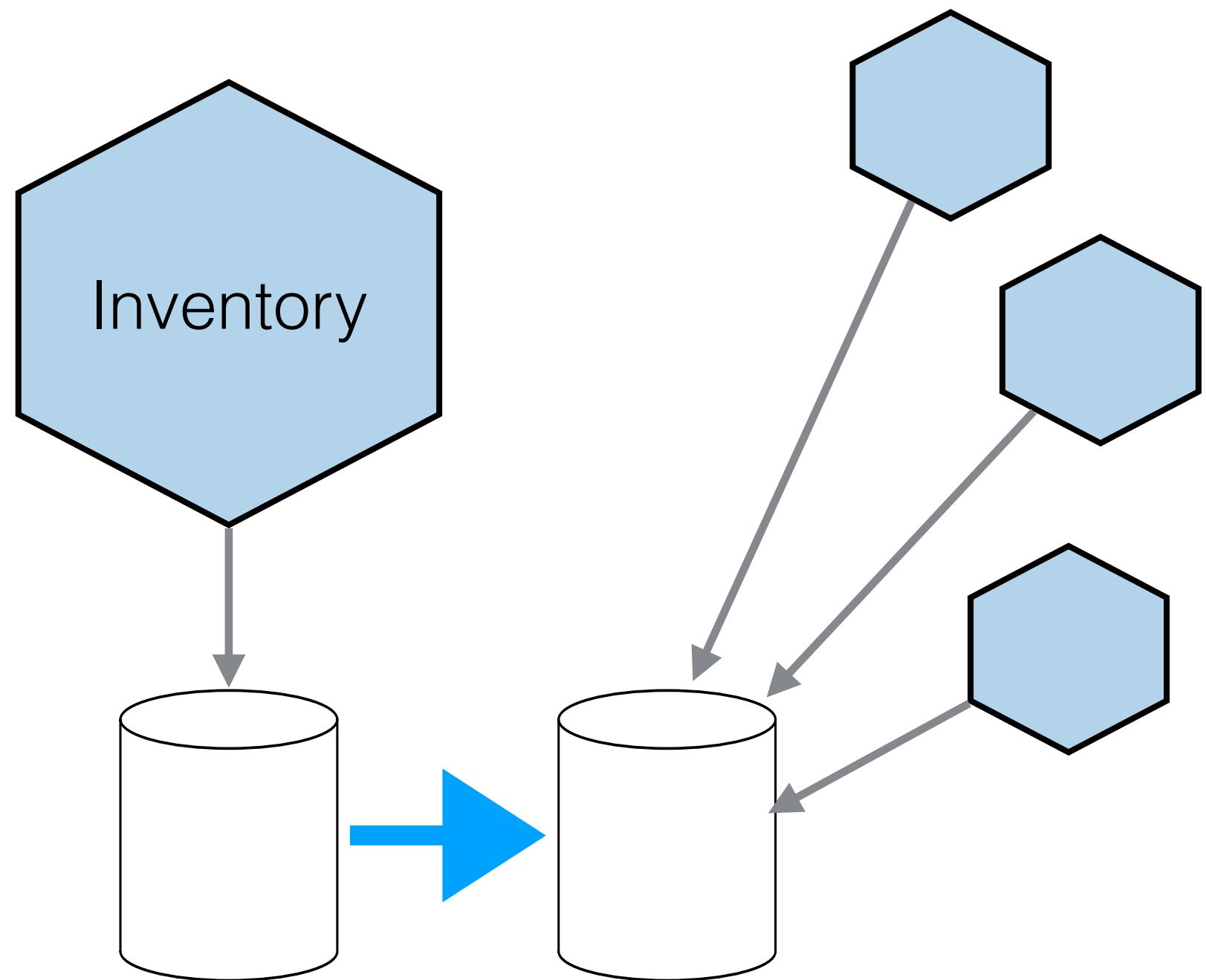


Allows for limited information hiding for direct DB access

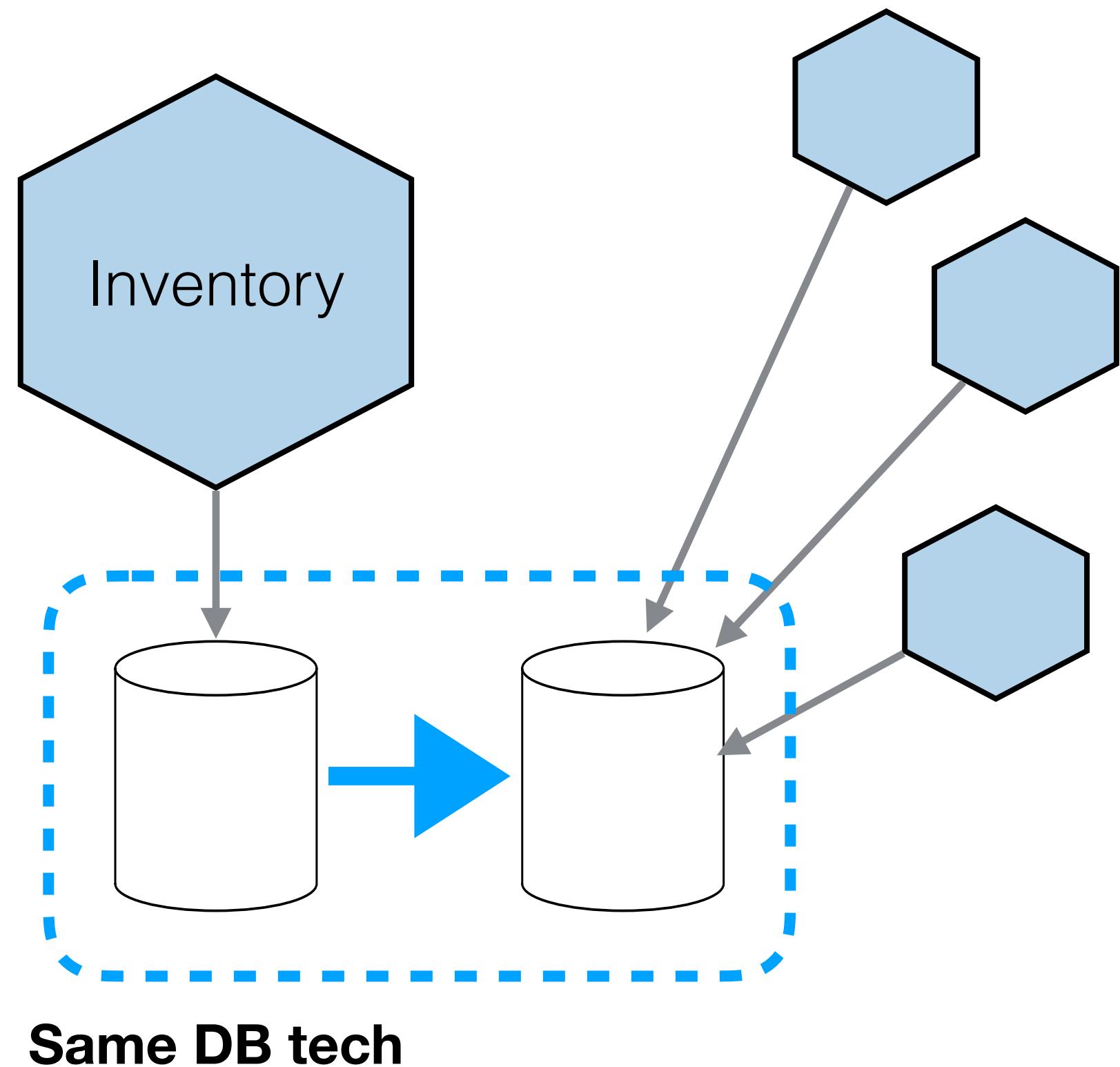
## DATABASE VIEW - PROS AND CONS



## DATABASE VIEW - PROS AND CONS

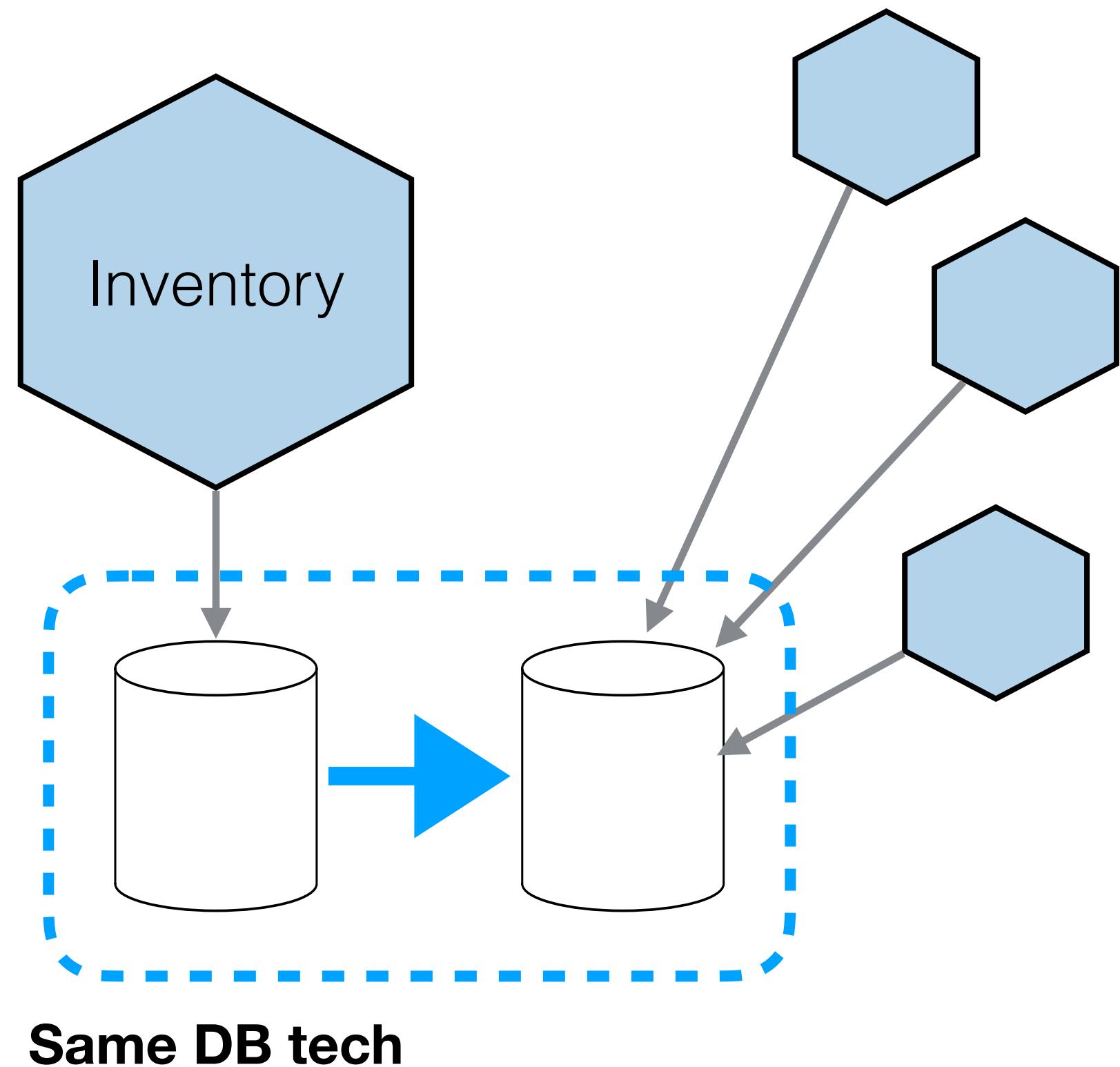


## DATABASE VIEW - PROS AND CONS



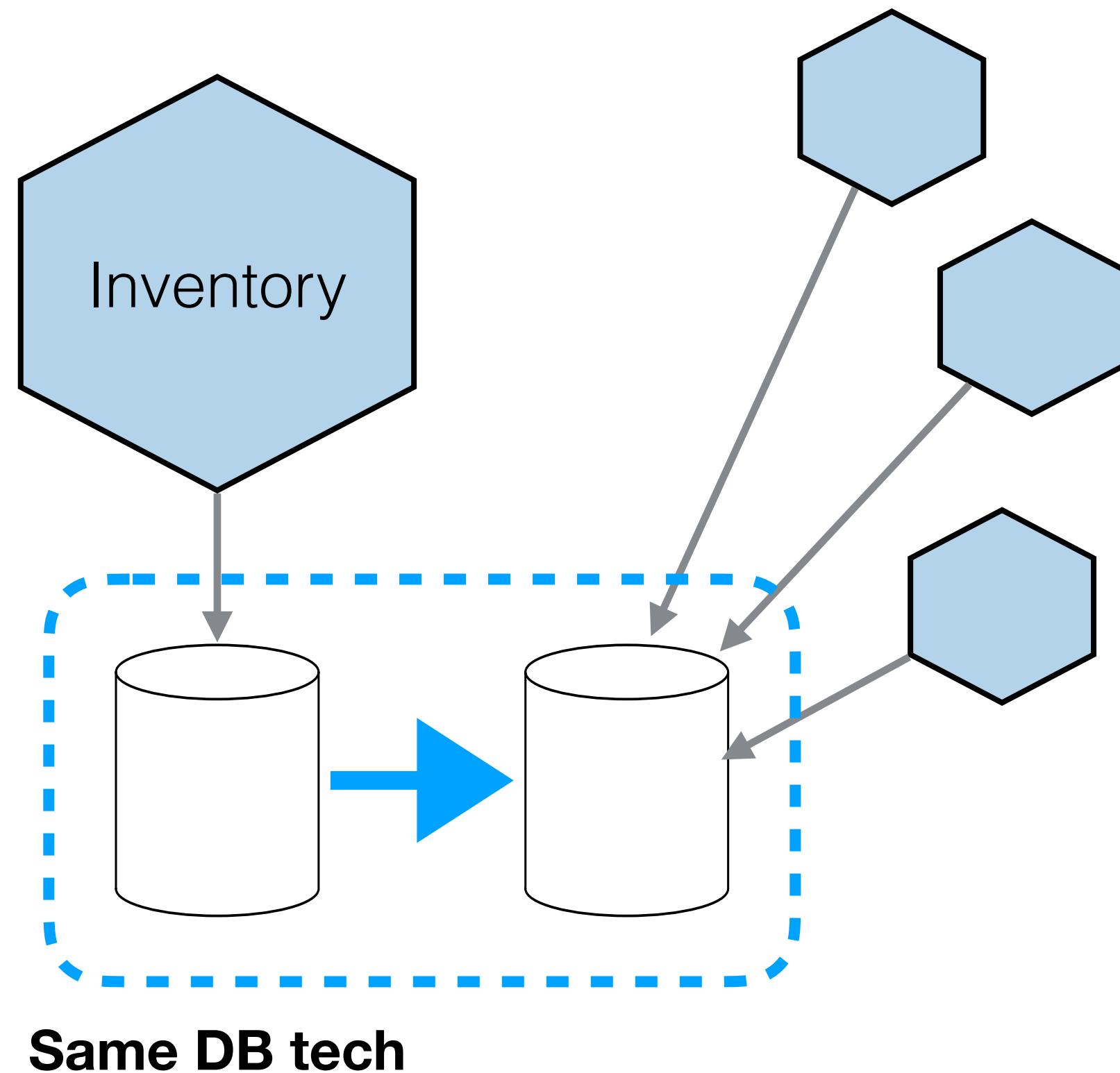
Tied to an underlying implementation

## DATABASE VIEW - PROS AND CONS



Tied to an underlying implementation  
Doesn't solve writes

## DATABASE VIEW - PROS AND CONS

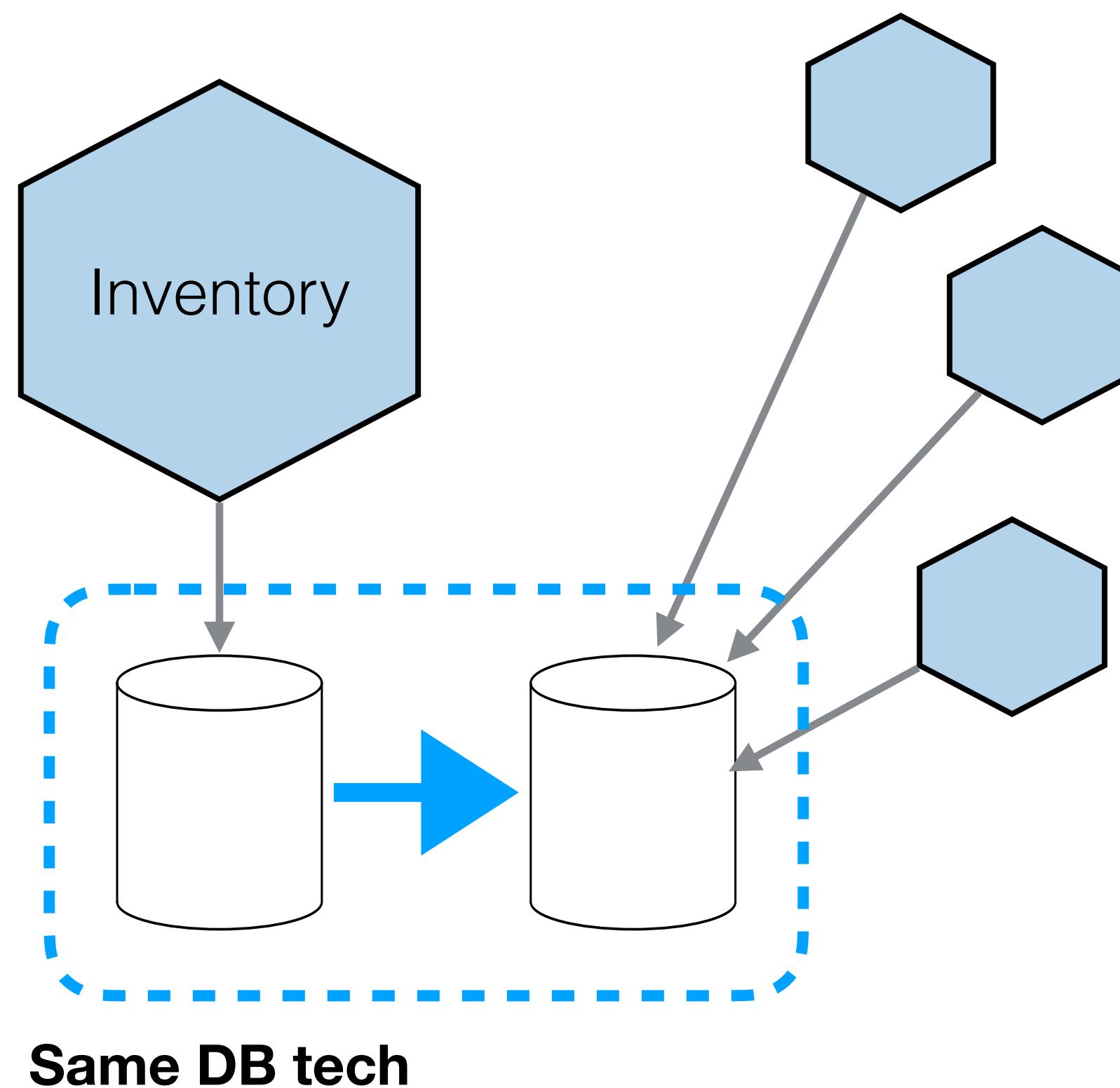


Tied to an underlying implementation

Doesn't solve writes

View mapping needs to be maintained if  
the source DB changes

## DATABASE VIEW - PROS AND CONS



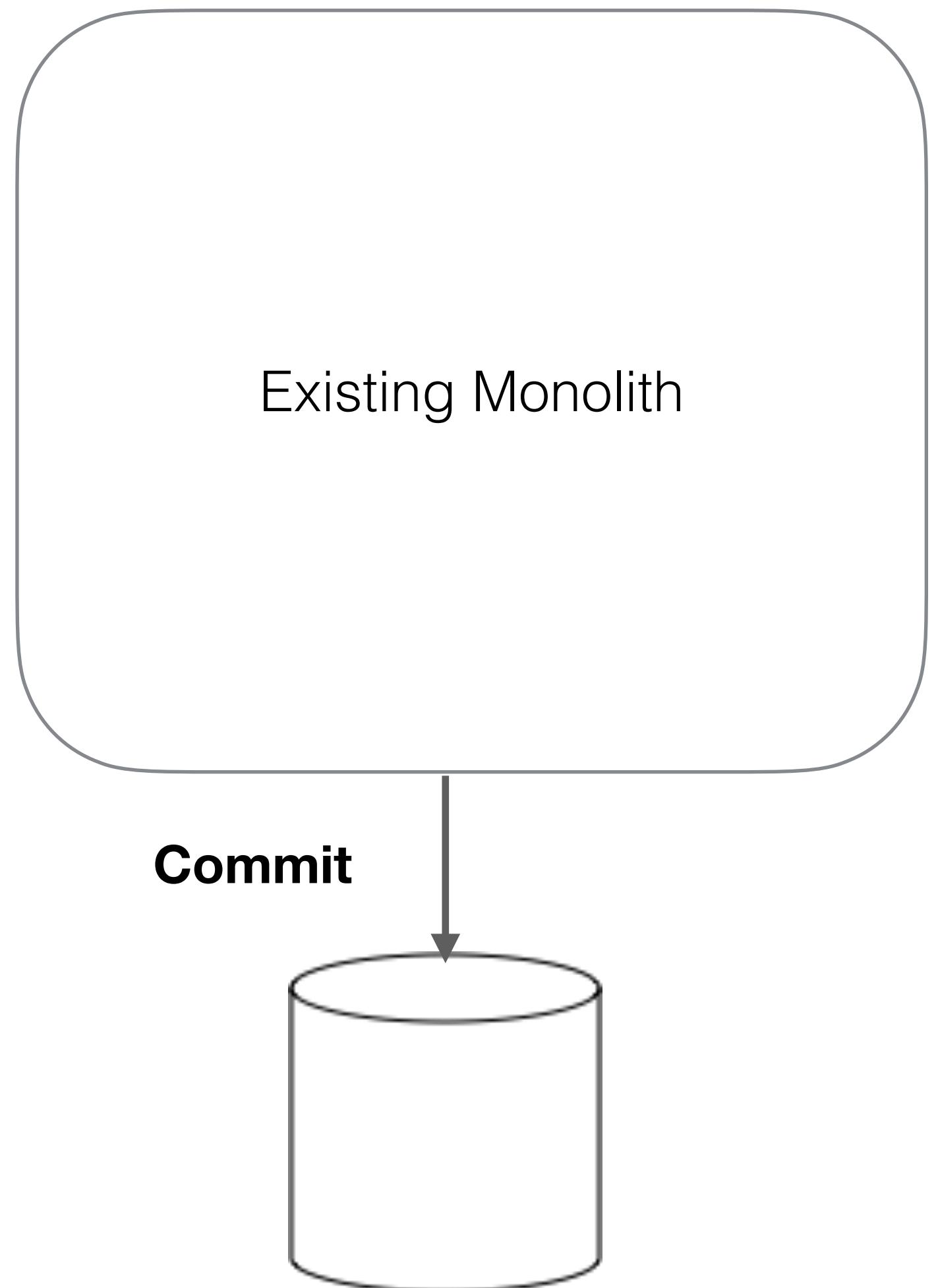
Tied to an underlying implementation

Doesn't solve writes

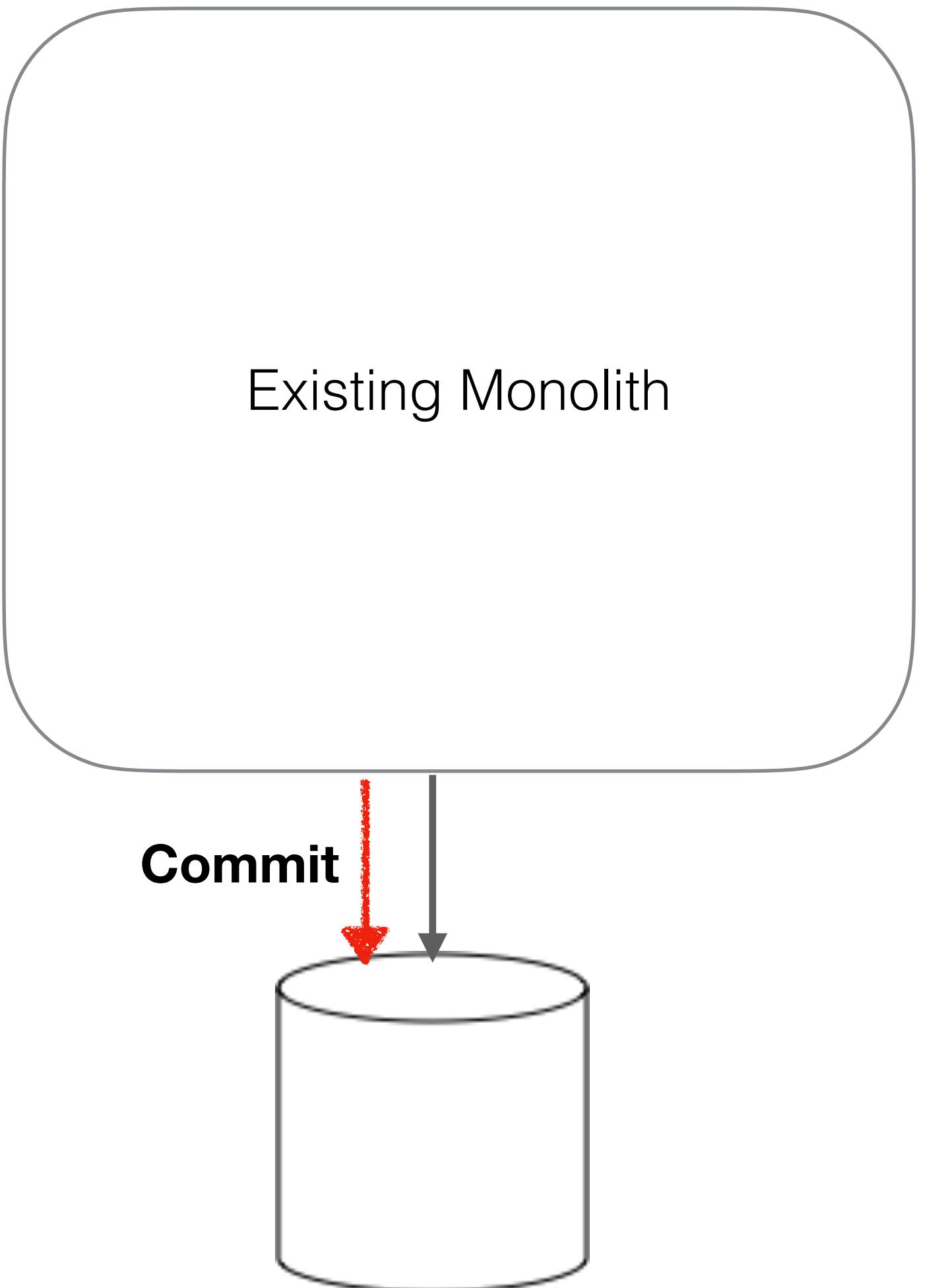
View mapping needs to be maintained if  
the source DB changes

Better than direct DB access

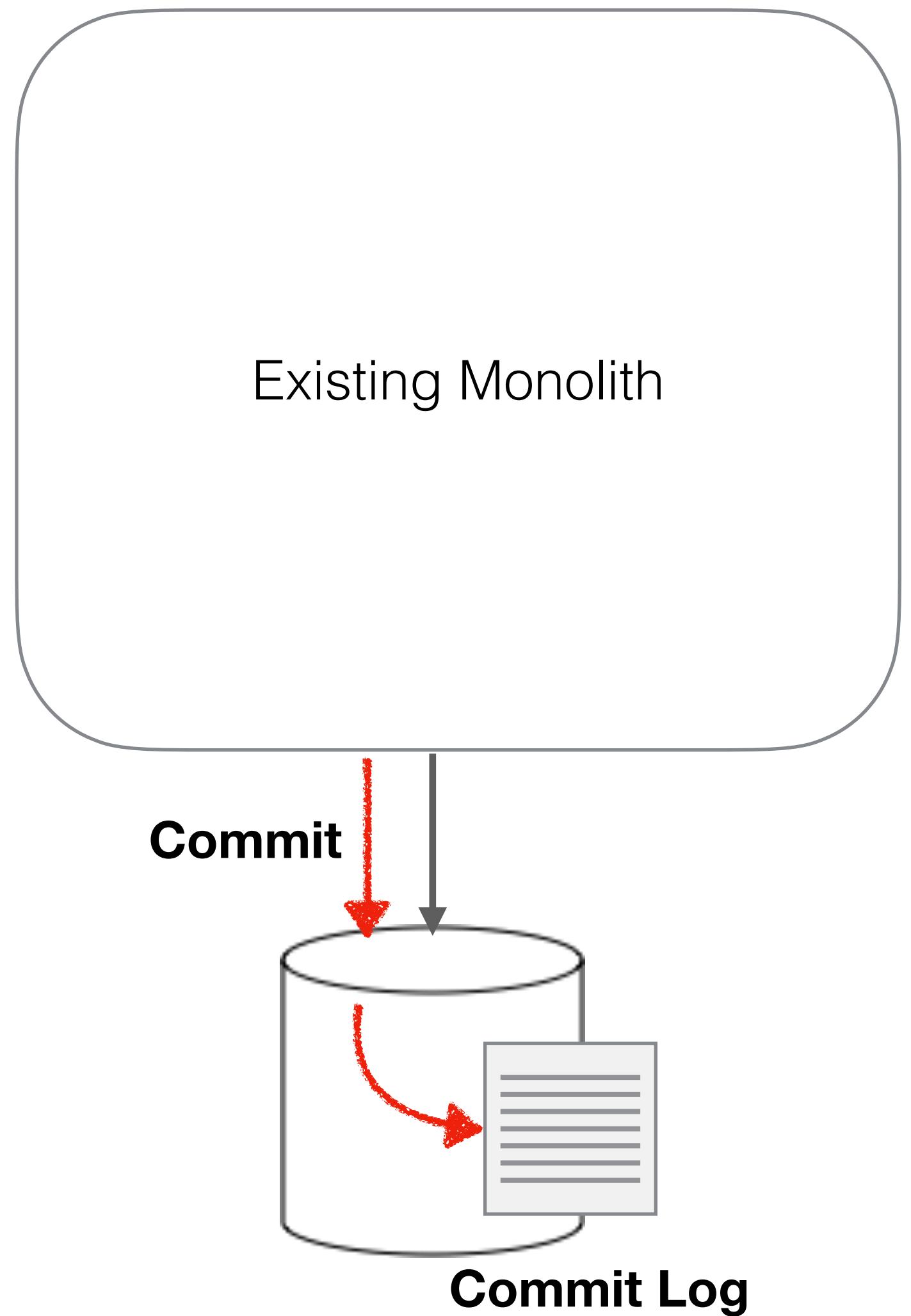
## PATTERN: CHANGE DATA CAPTURE



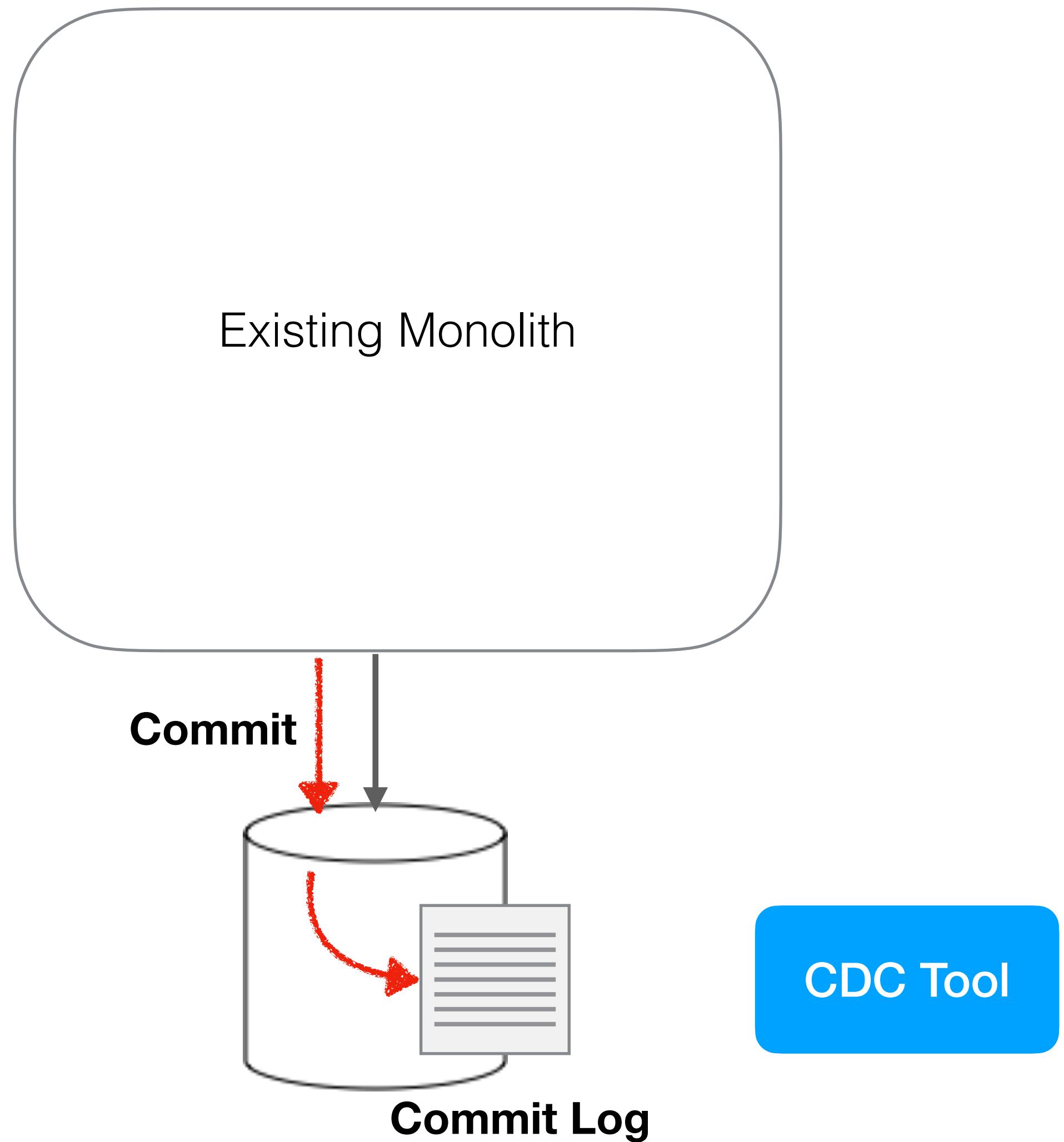
## PATTERN: CHANGE DATA CAPTURE



## PATTERN: CHANGE DATA CAPTURE

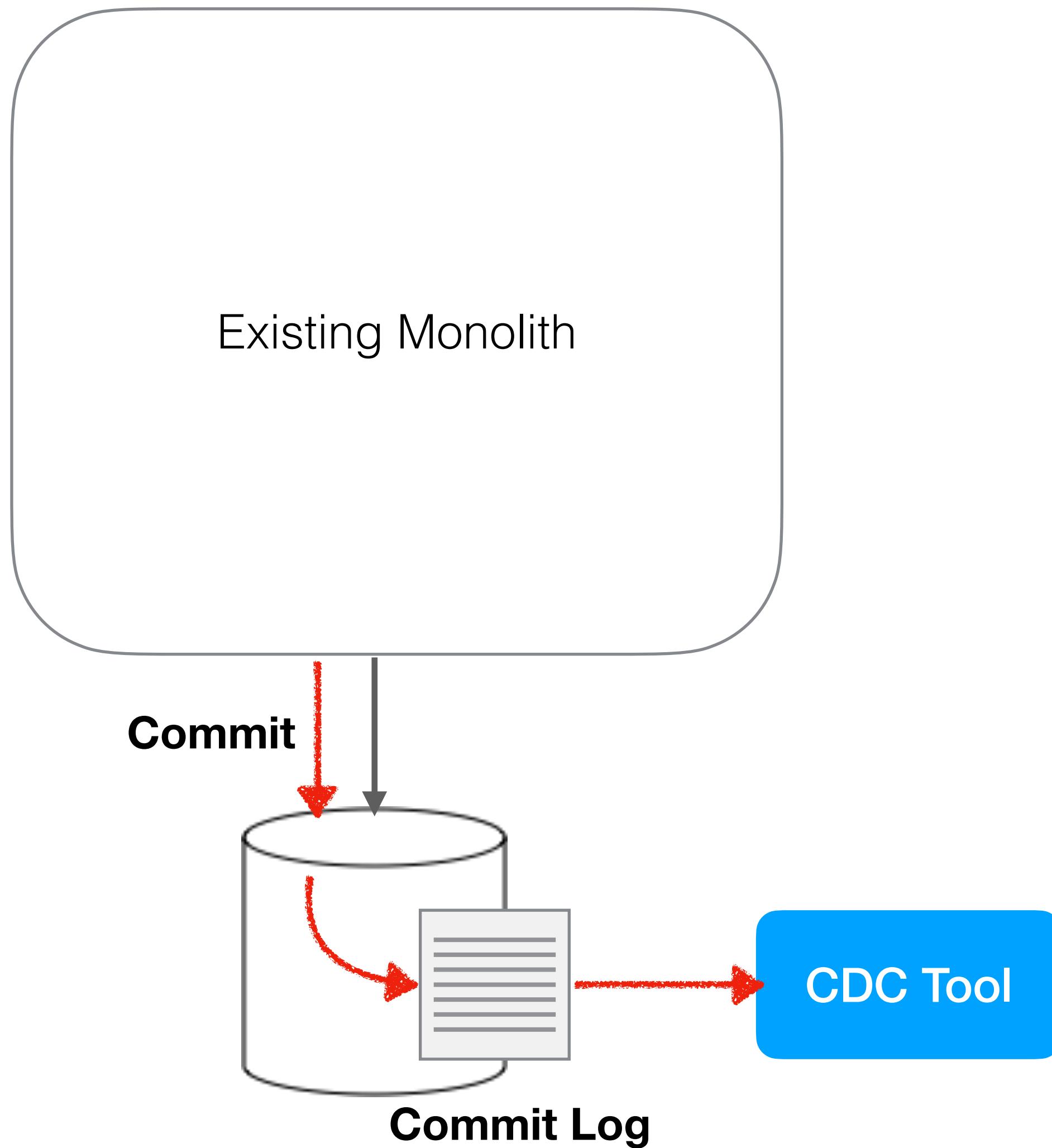


## PATTERN: CHANGE DATA CAPTURE

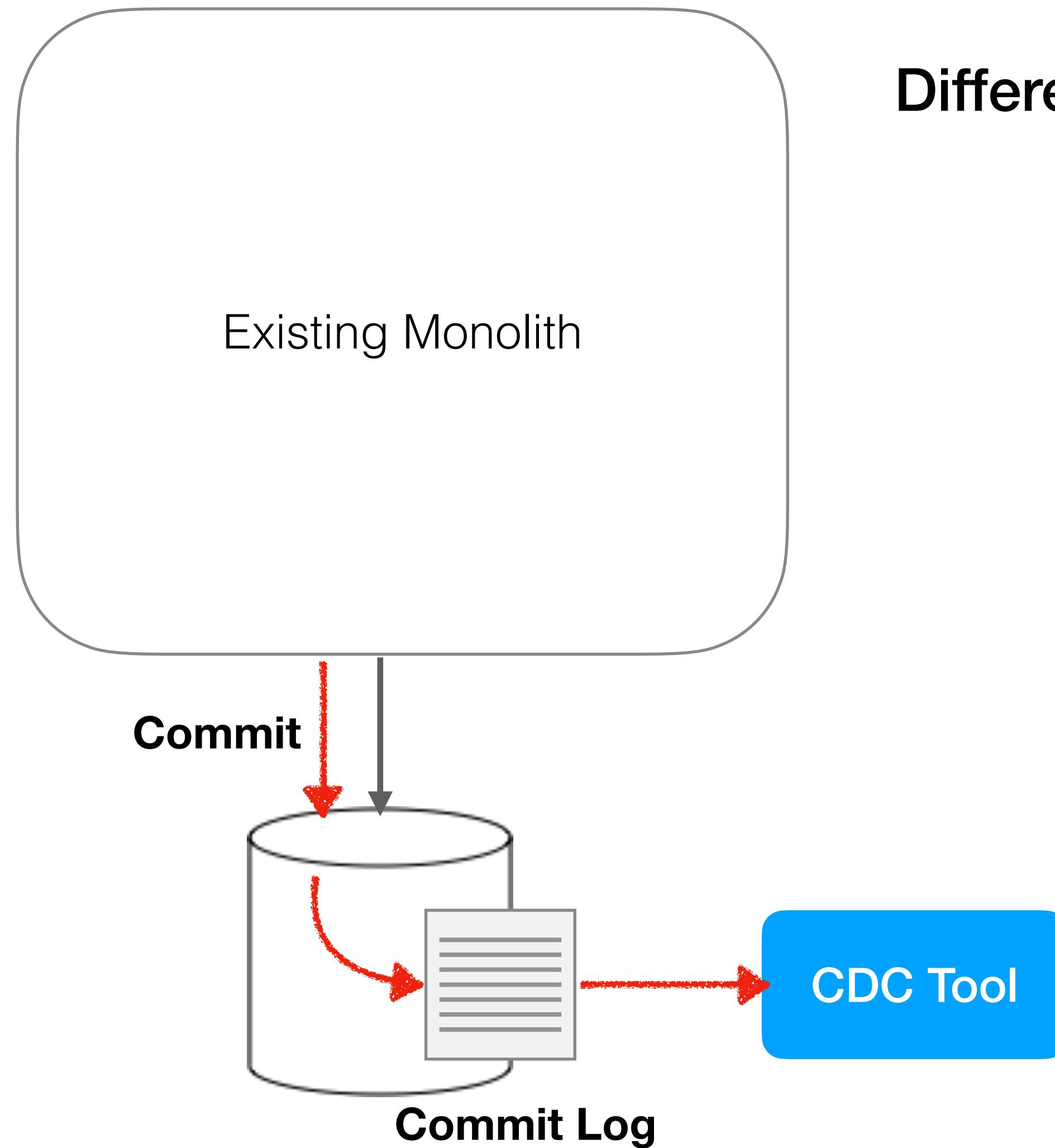


## PATTERN: CHANGE DATA CAPTURE

Capture data as its inserted



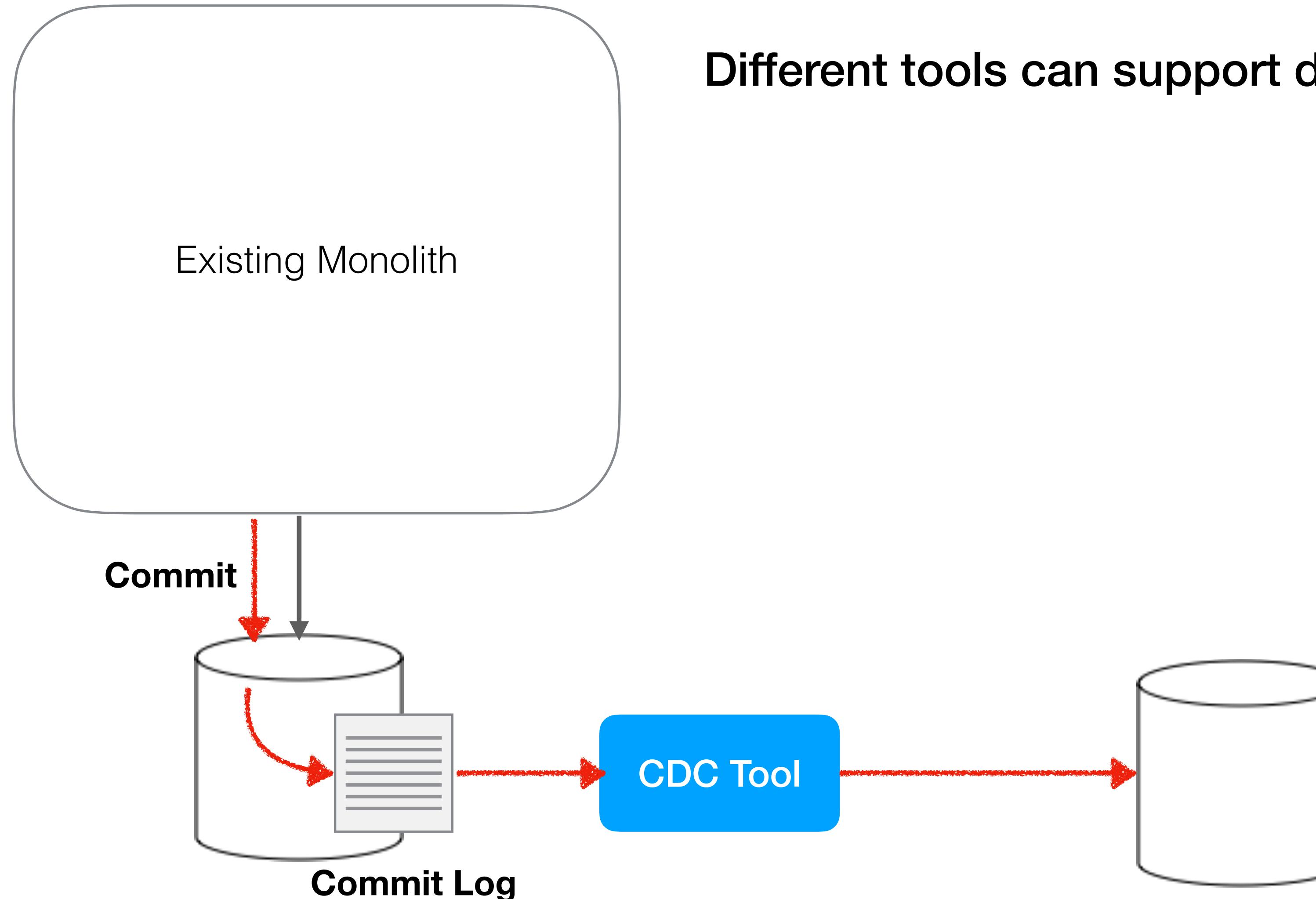
## PATTERN: CHANGE DATA CAPTURE



Capture data as its inserted

Different tools can support different destinations

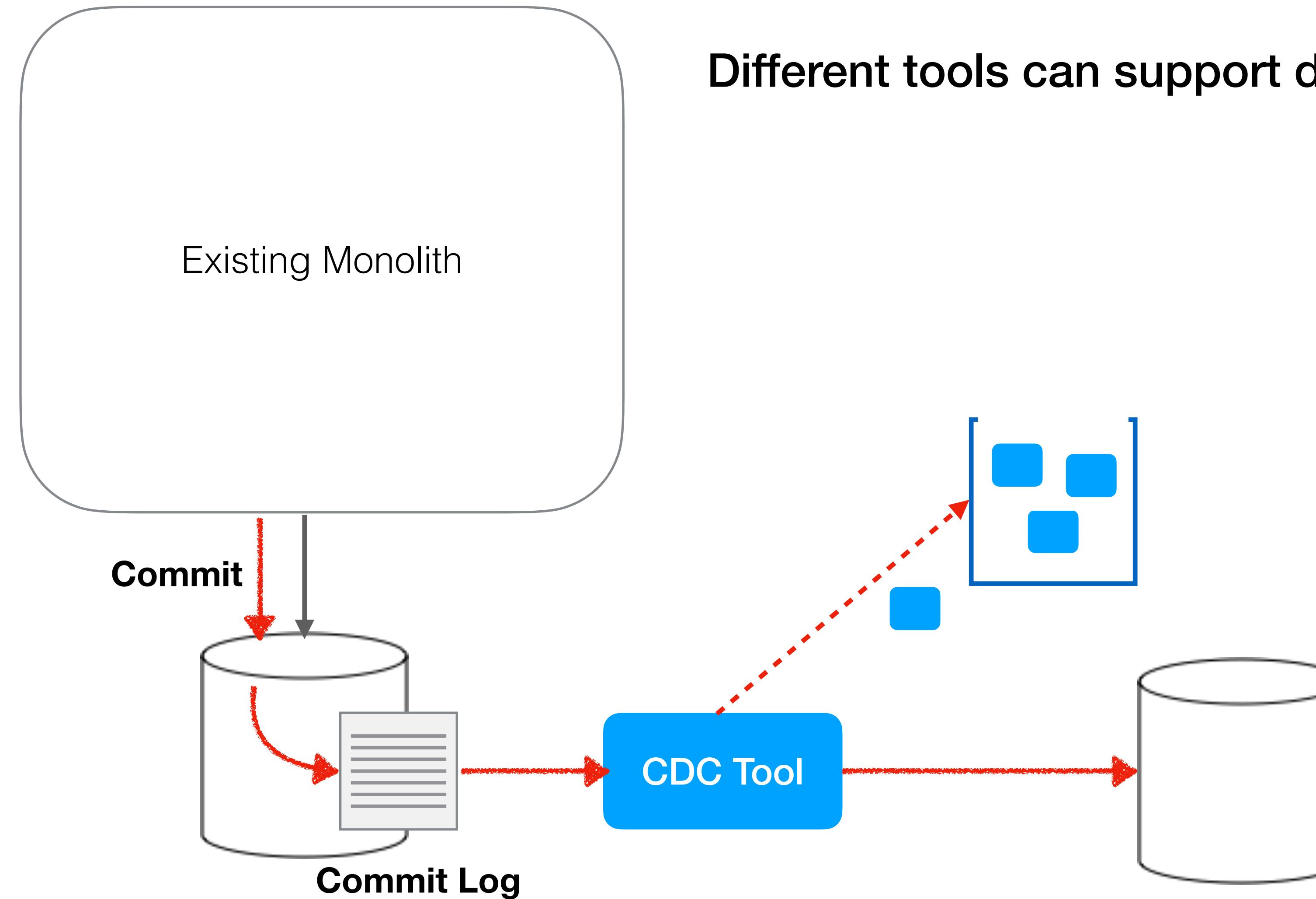
## PATTERN: CHANGE DATA CAPTURE



Capture data as its inserted

Different tools can support different destinations

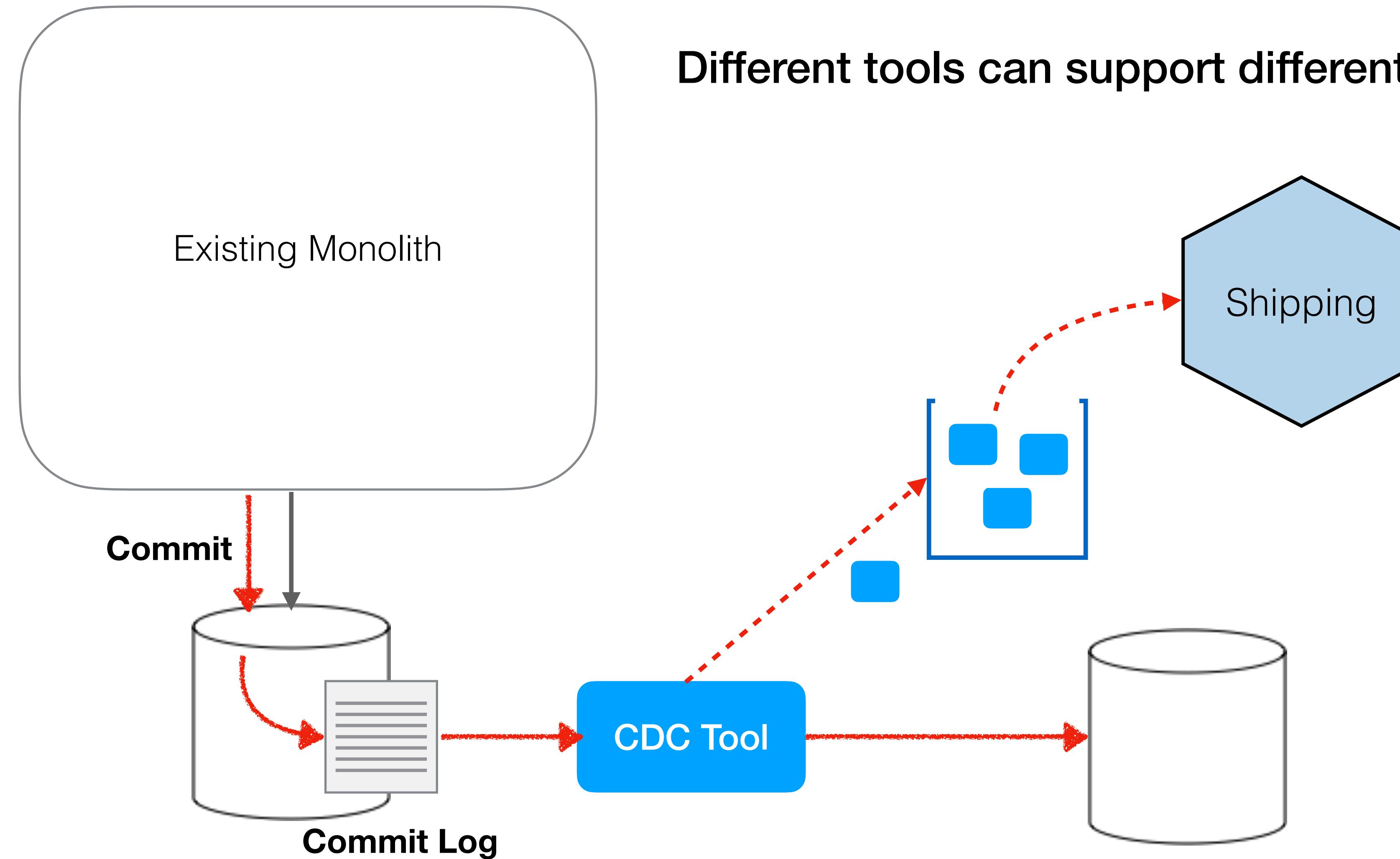
## PATTERN: CHANGE DATA CAPTURE



Capture data as its inserted

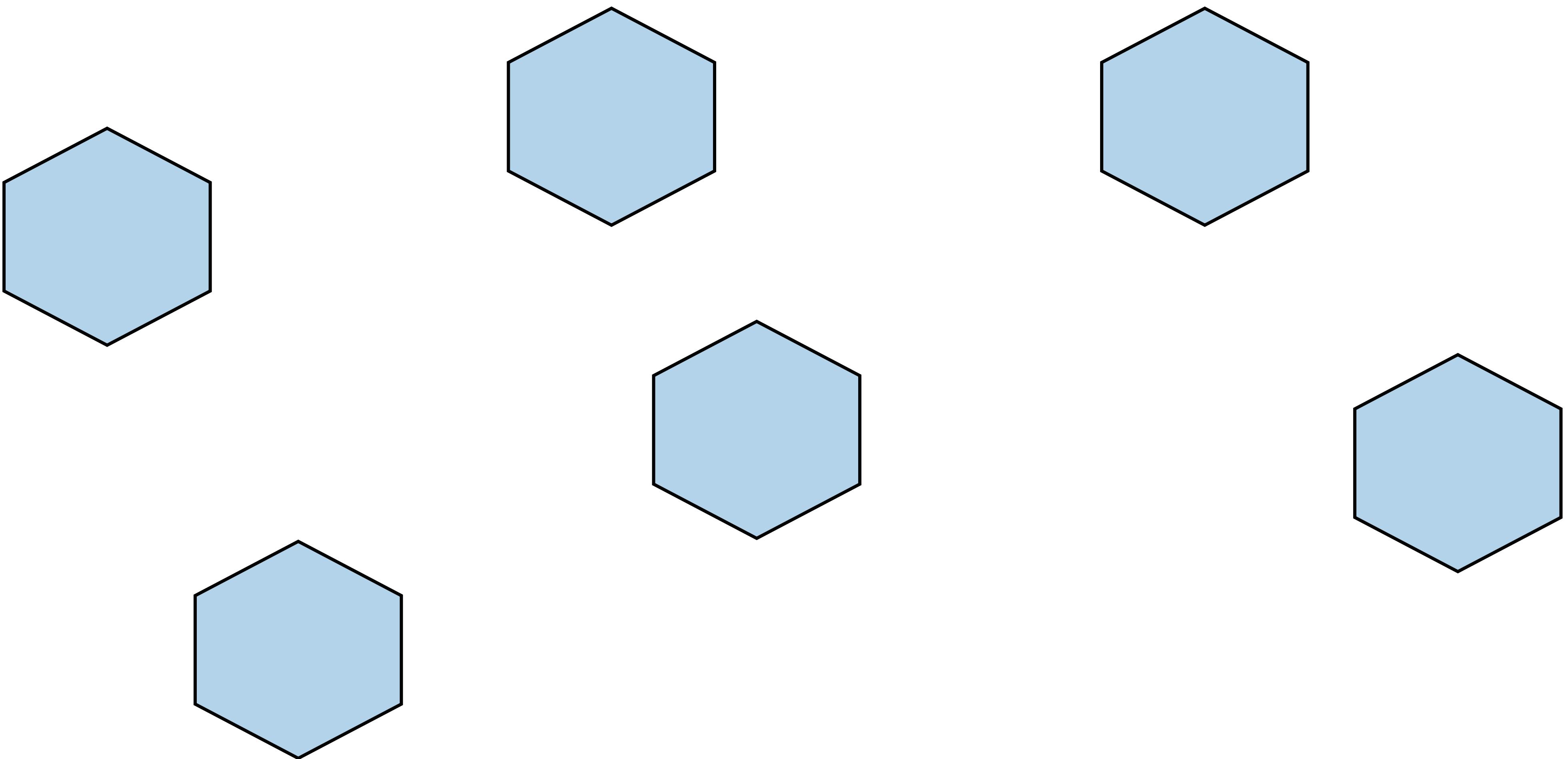
Different tools can support different destinations

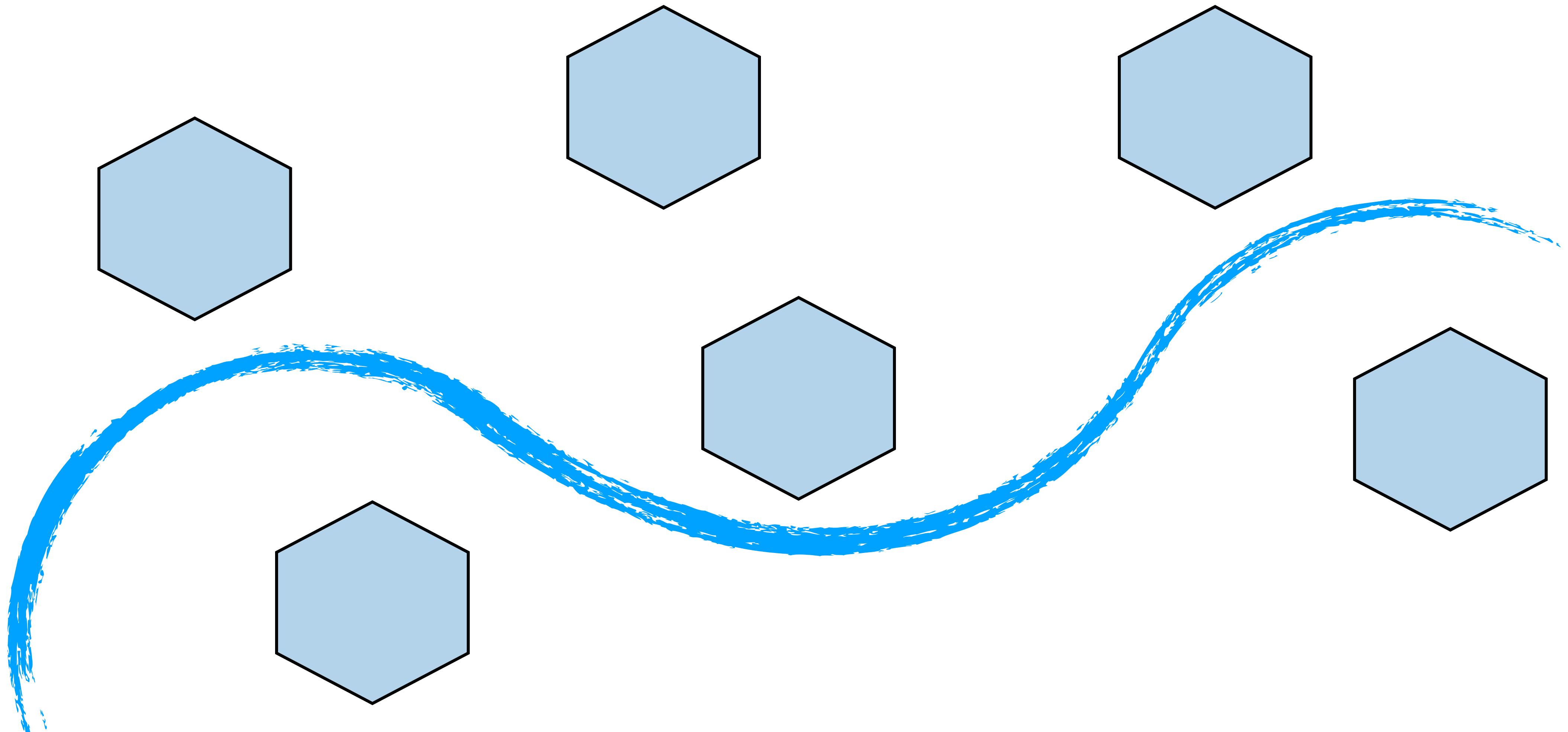
## PATTERN: CHANGE DATA CAPTURE

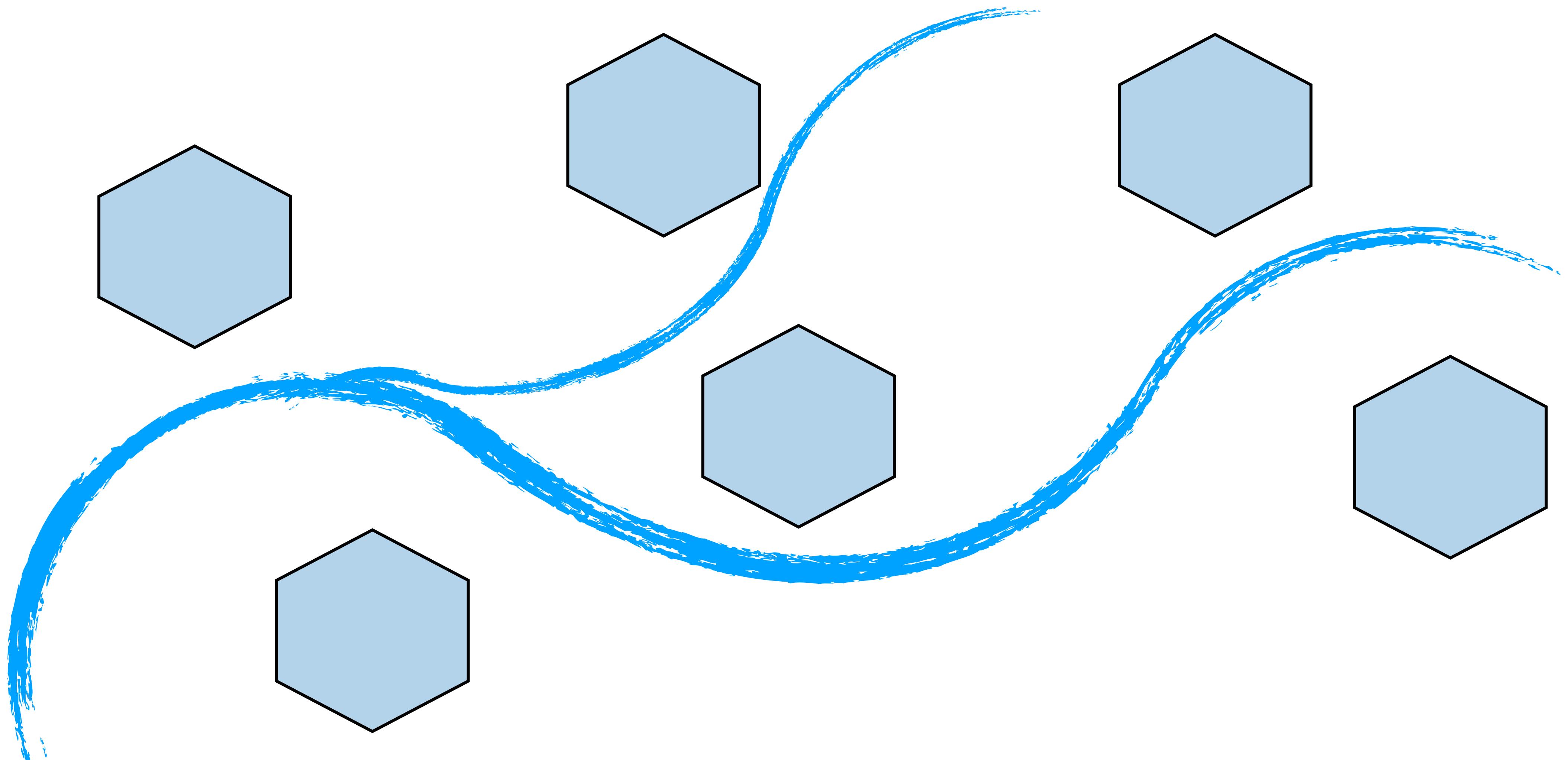


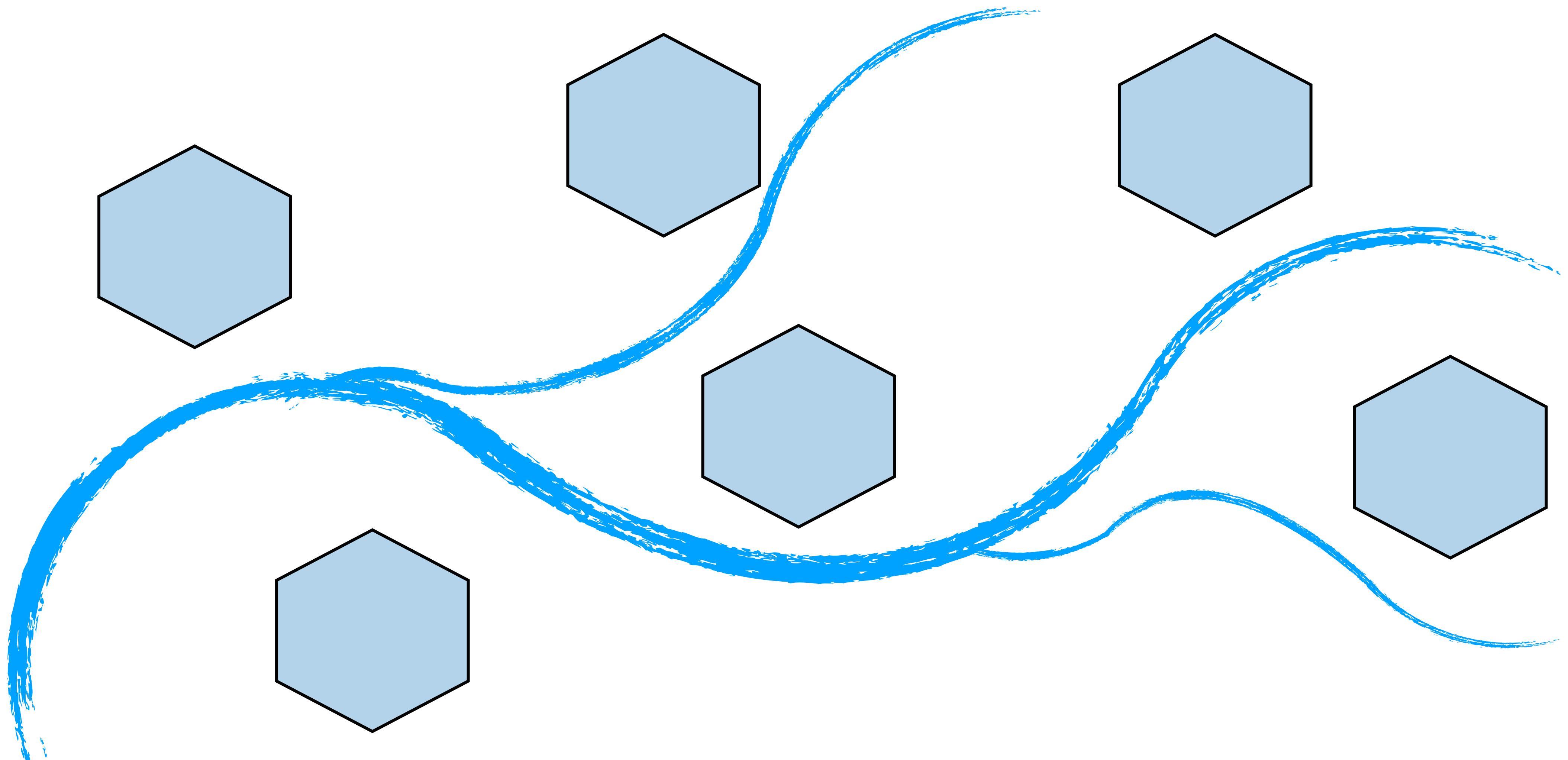
Capture data as its inserted

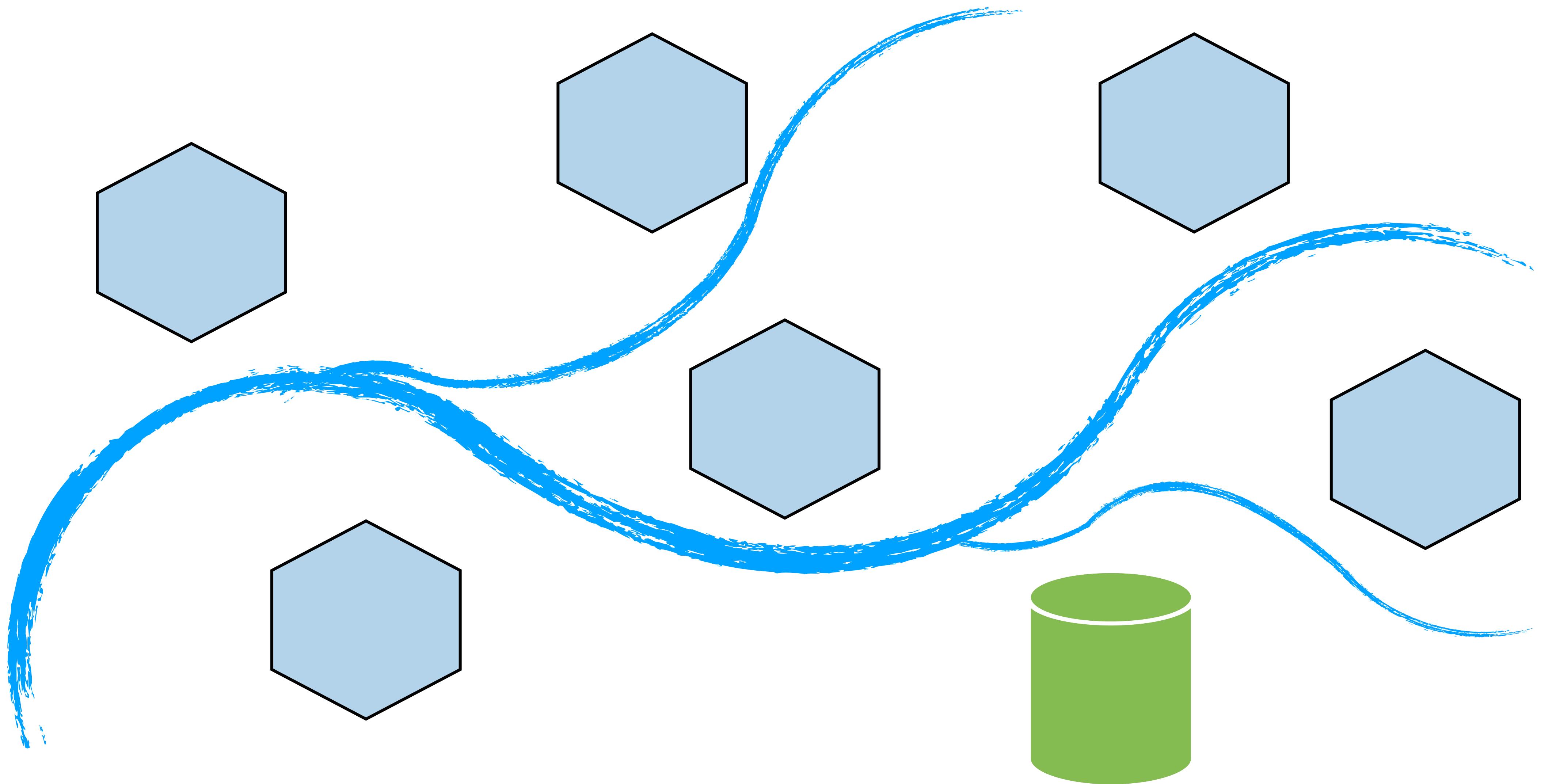
Different tools can support different destinations

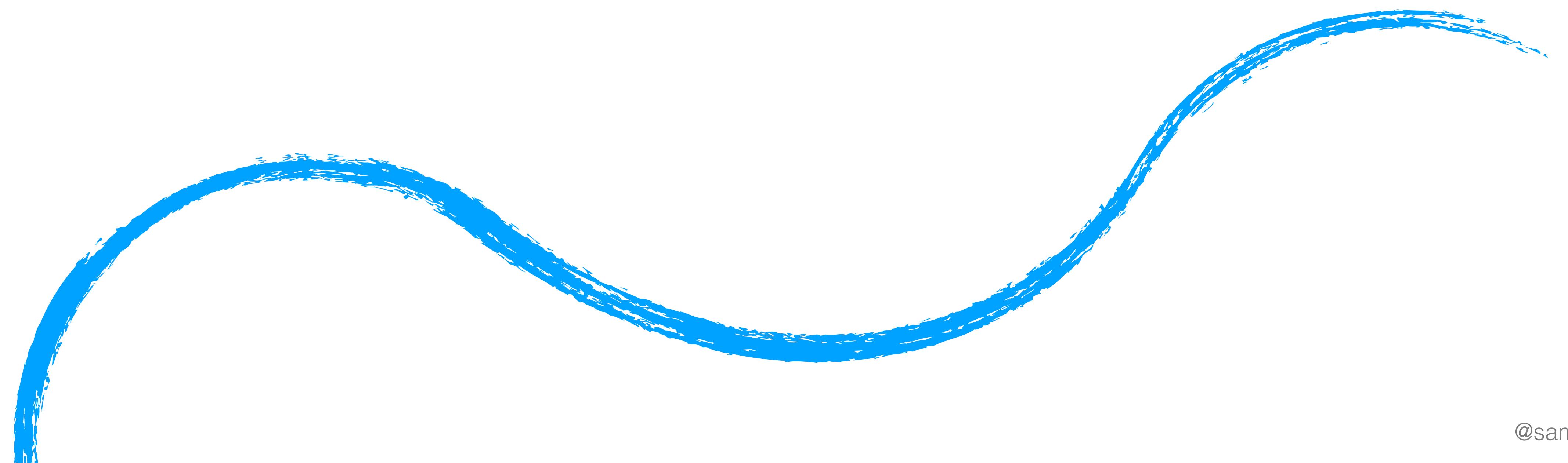
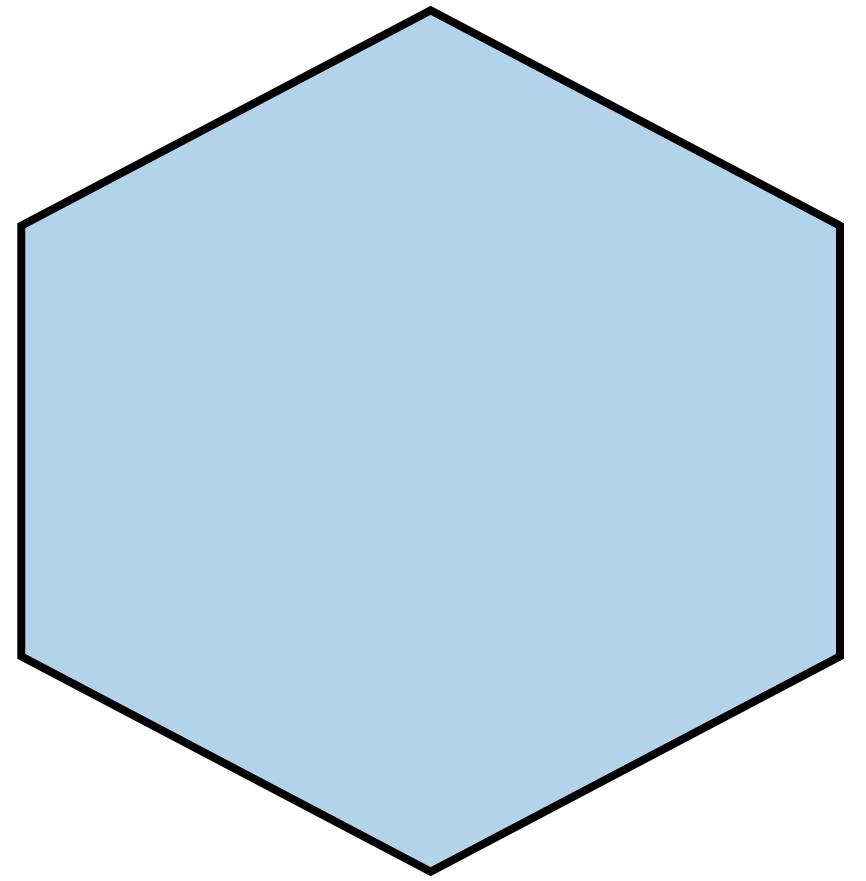




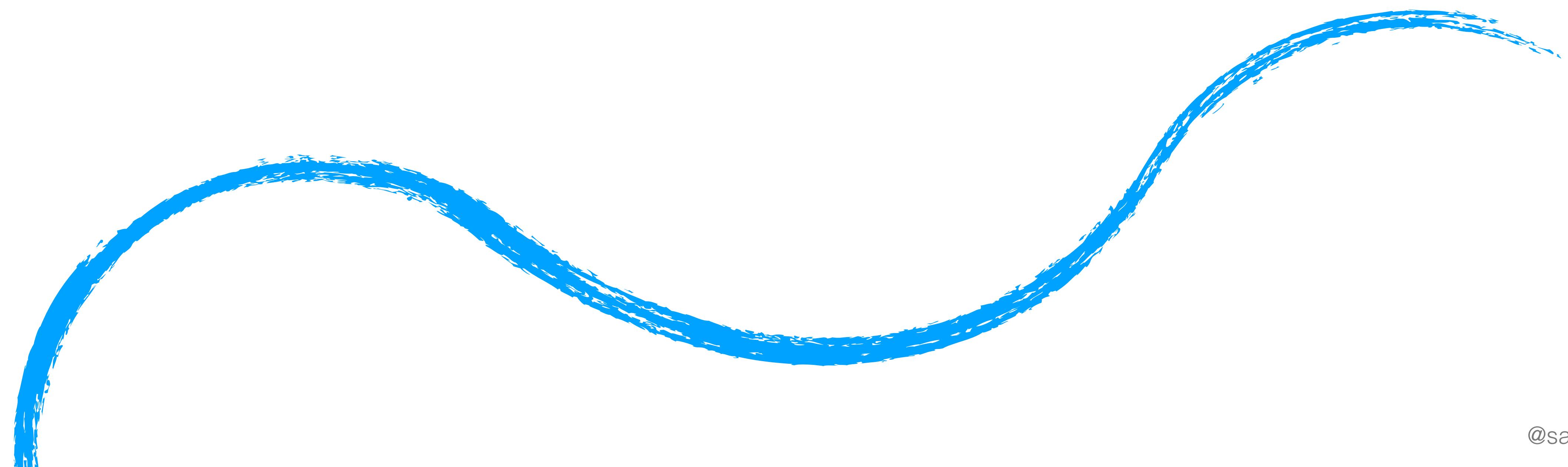
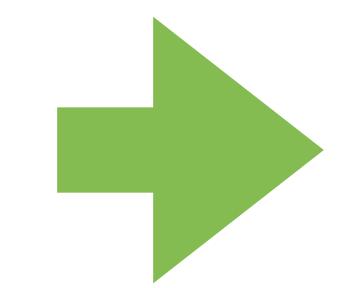
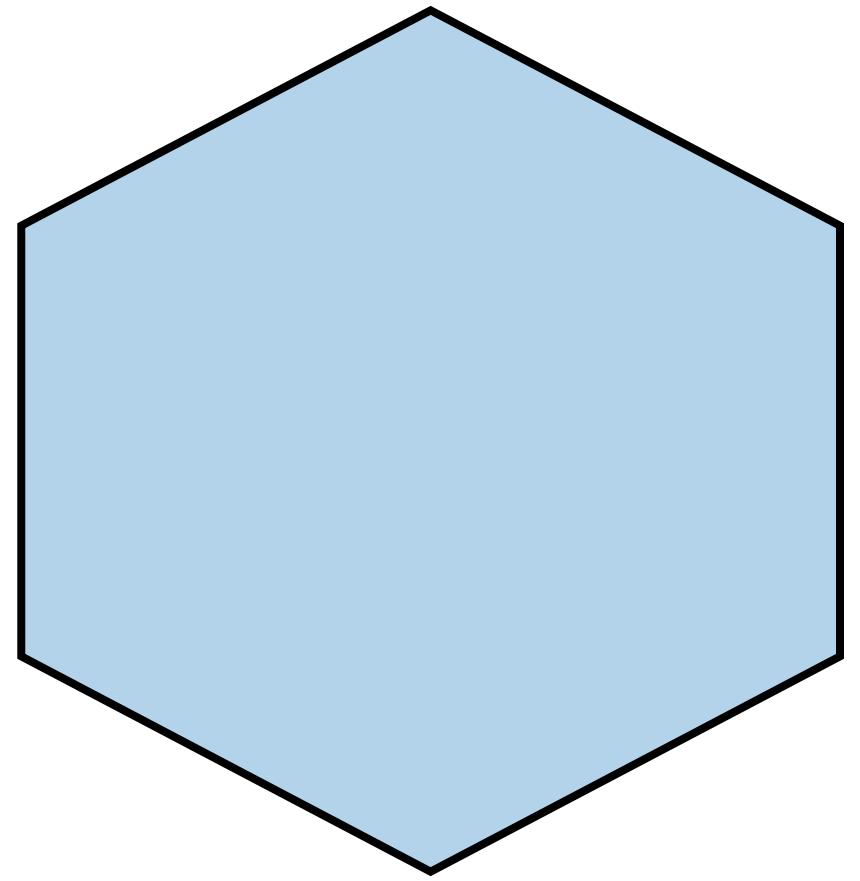






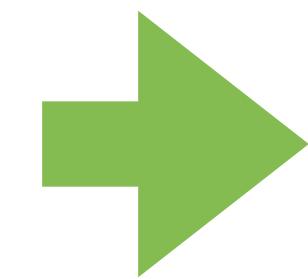
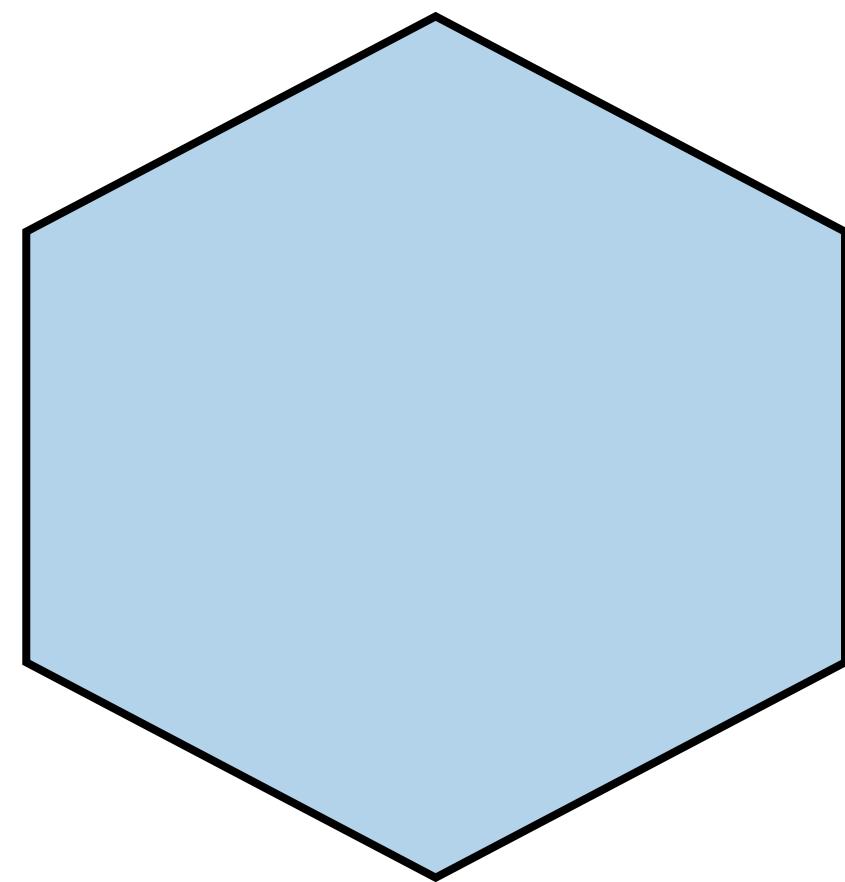


@samnewman

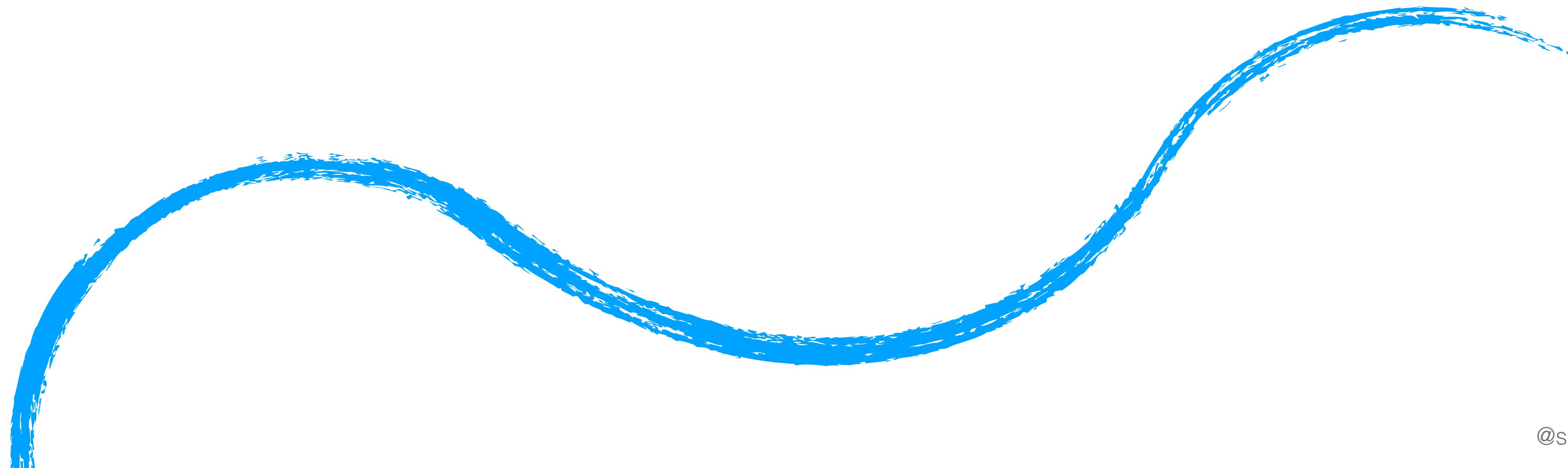


@samnewman

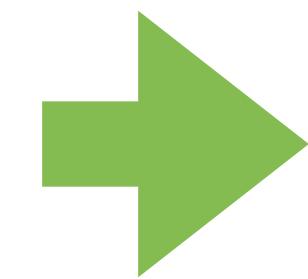
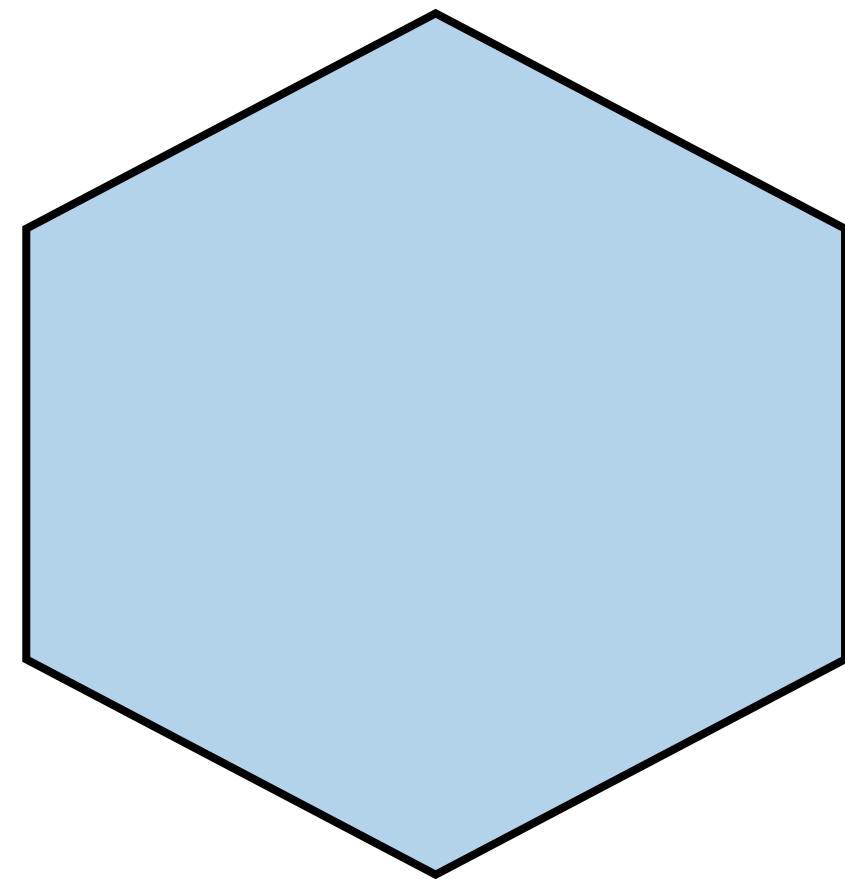
**Event**



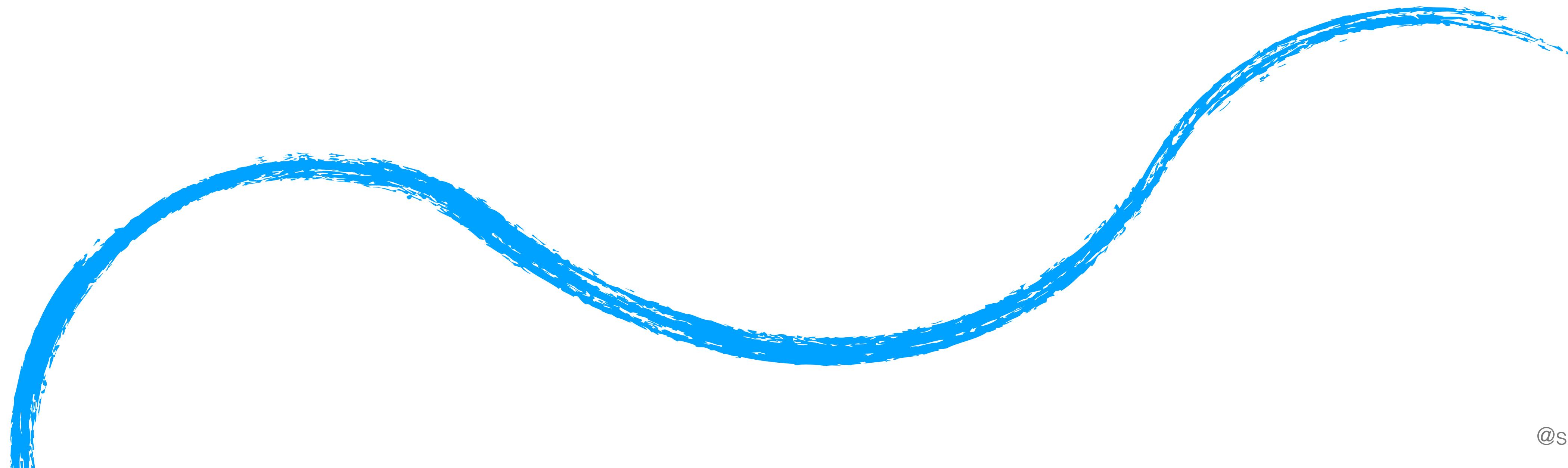
**Order Updated!**



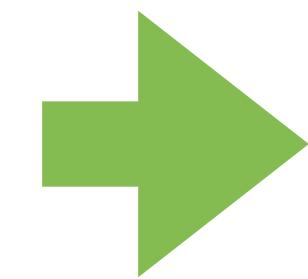
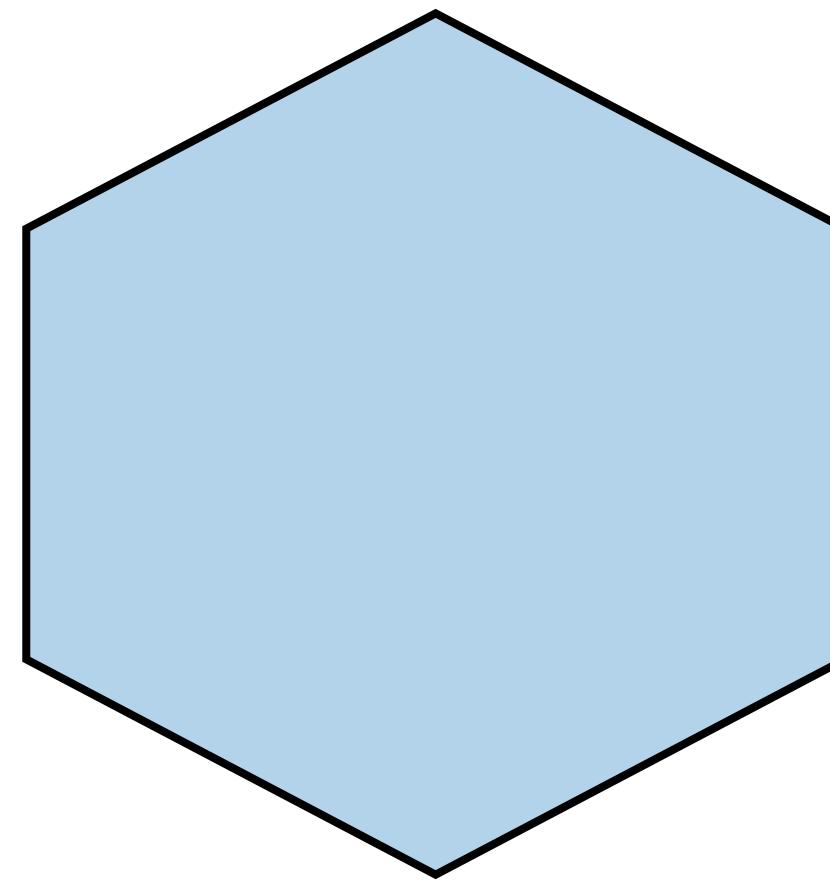
# Event



**Order Updated!**  
**04/08/2020**



# Event

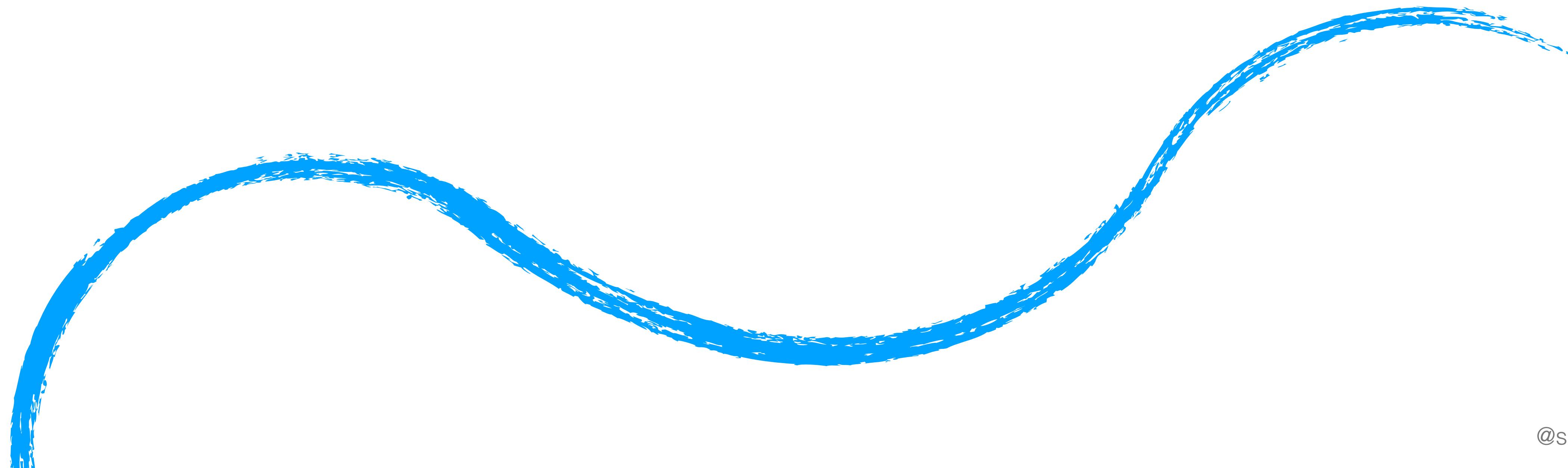


**Order Updated!**

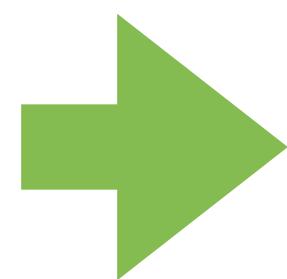
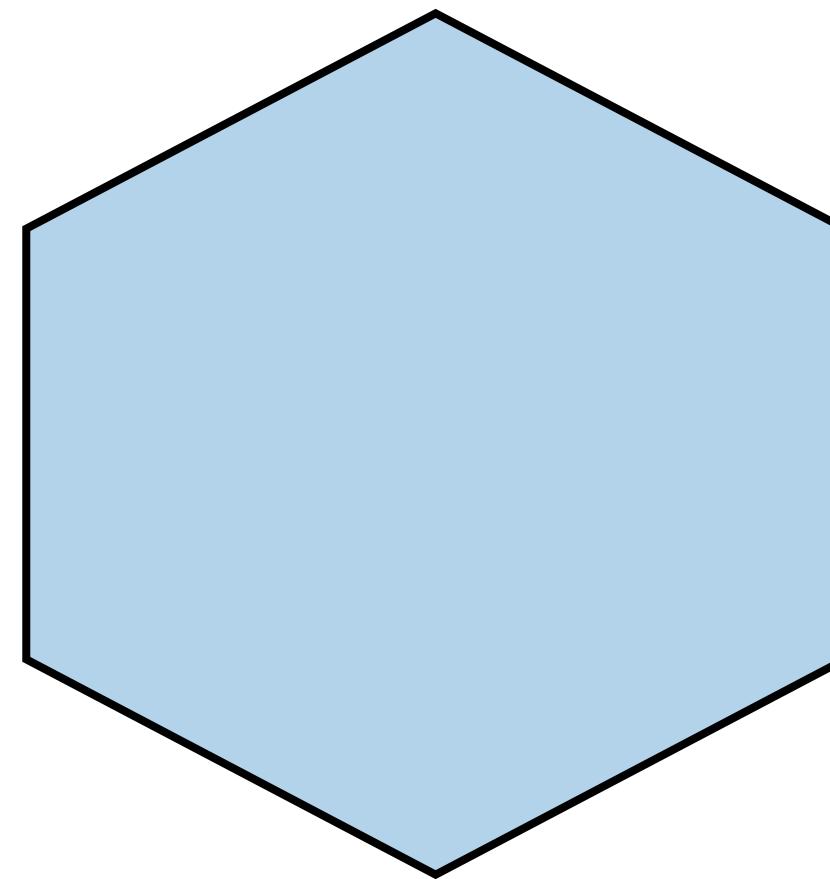
**04/08/2020**

**Customer: Sam Newman**

**Order Value: \$130.25**



# Event

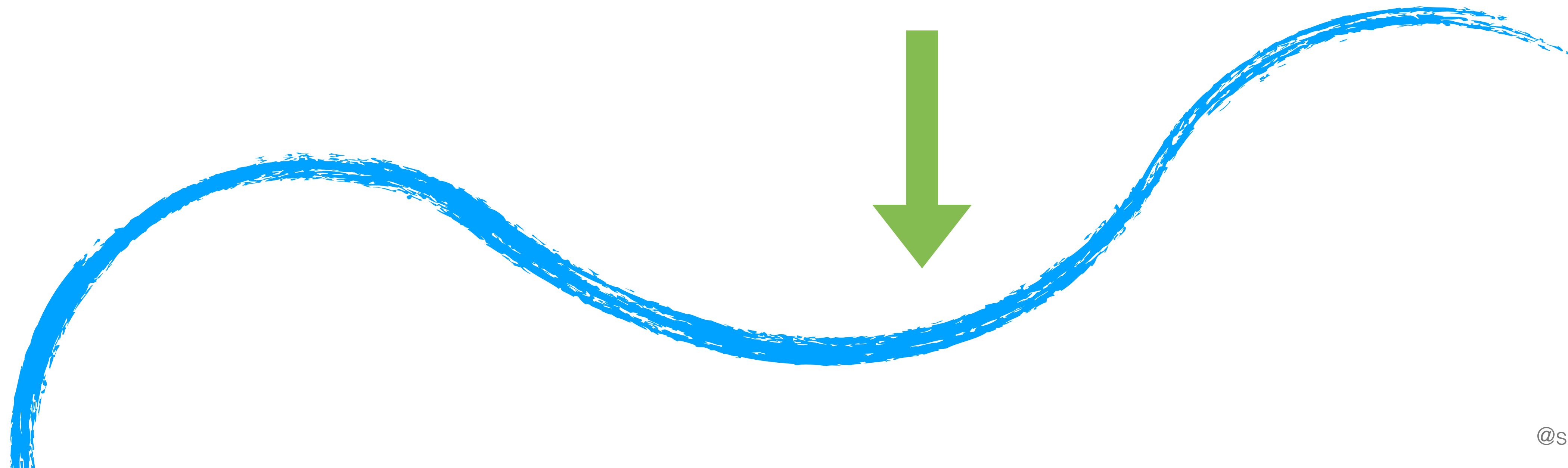


**Order Updated!**

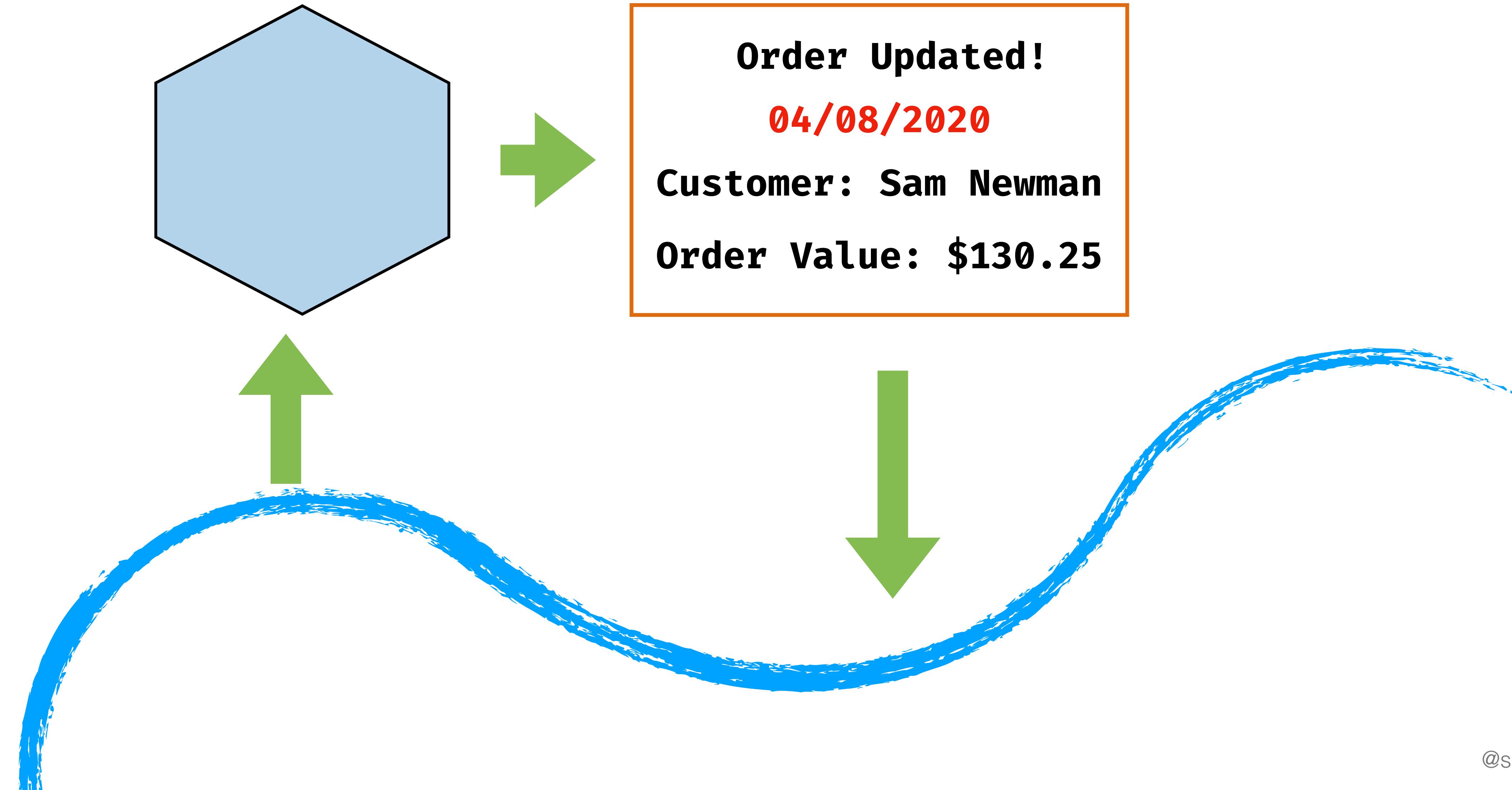
**04/08/2020**

**Customer: Sam Newman**

**Order Value: \$130.25**



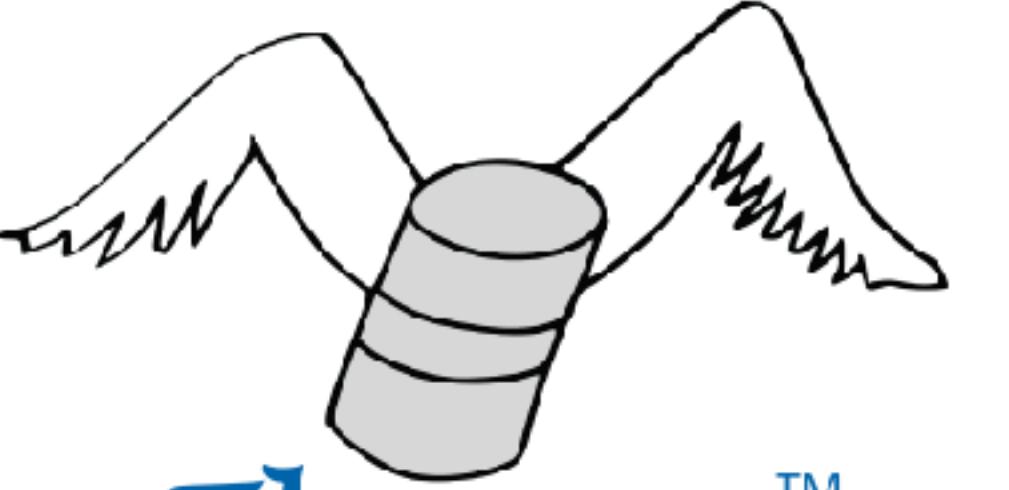
# Event



# **4/4 Data Decomposition**

# DATA REFACTORING TOOLS

GET STARTED   DOWNLOAD / PRICING   DOCUMENTATION   BLOG



**Flyway™**  
by Redgate

**Version control for your database.**  
**Robust schema evolution across all your environments.**  
**With ease, pleasure and plain SQL.**

[Get Started with Flyway 6.4.4 →](#)

[Download](#) - [Release Notes](#) - [Apache v2 license](#)  
Works on:      

Stay updated with our newsletter

<https://flywaydb.org/>

@samnewman

# DATA REFACTORING TOOLS

The Flyway logo consists of a stylized illustration of a grey barrel resting on a black, wavy line that resembles both a mountain range and a winding path. Below the illustration, the word "Flyway" is written in a large, blue, lowercase, sans-serif font. A small "TM" symbol is positioned at the top right of the "y". Underneath "Flyway", the words "by Redgate" are written in a smaller, grey, lowercase, sans-serif font.

**Each DB refactoring is stored in version control**

**https://flywaydb.org/**

@samnewman

## DATA REFACTORING TOOLS

The screenshot shows the homepage of the Flyway website. At the top, there is a navigation bar with links for "GET STARTED", "DOWNLOAD / PRICING", "DOCUMENTATION", and "BLOG". Below the navigation bar is the Flyway logo, which features a stylized grey cylinder (resembling a database) positioned between two wavy lines that suggest mountains or waves. The word "Flyway" is written in a blue, lowercase, sans-serif font next to the cylinder, with a trademark symbol (TM) at the end. Below the logo, the text "by Redgate" is visible. The main headline reads "Version control for your database. Robust schema evolution across all your environments. With ease, pleasure and plain SQL." Below this headline is a blue call-to-action button with white text that says "Get Started with Flyway 6.4.4 →". At the bottom of the page, there is a link to "Download - Release Notes - Apache v2 license" and a section titled "Works on:" followed by icons for various operating systems and databases. A newsletter sign-up form is also present at the bottom.

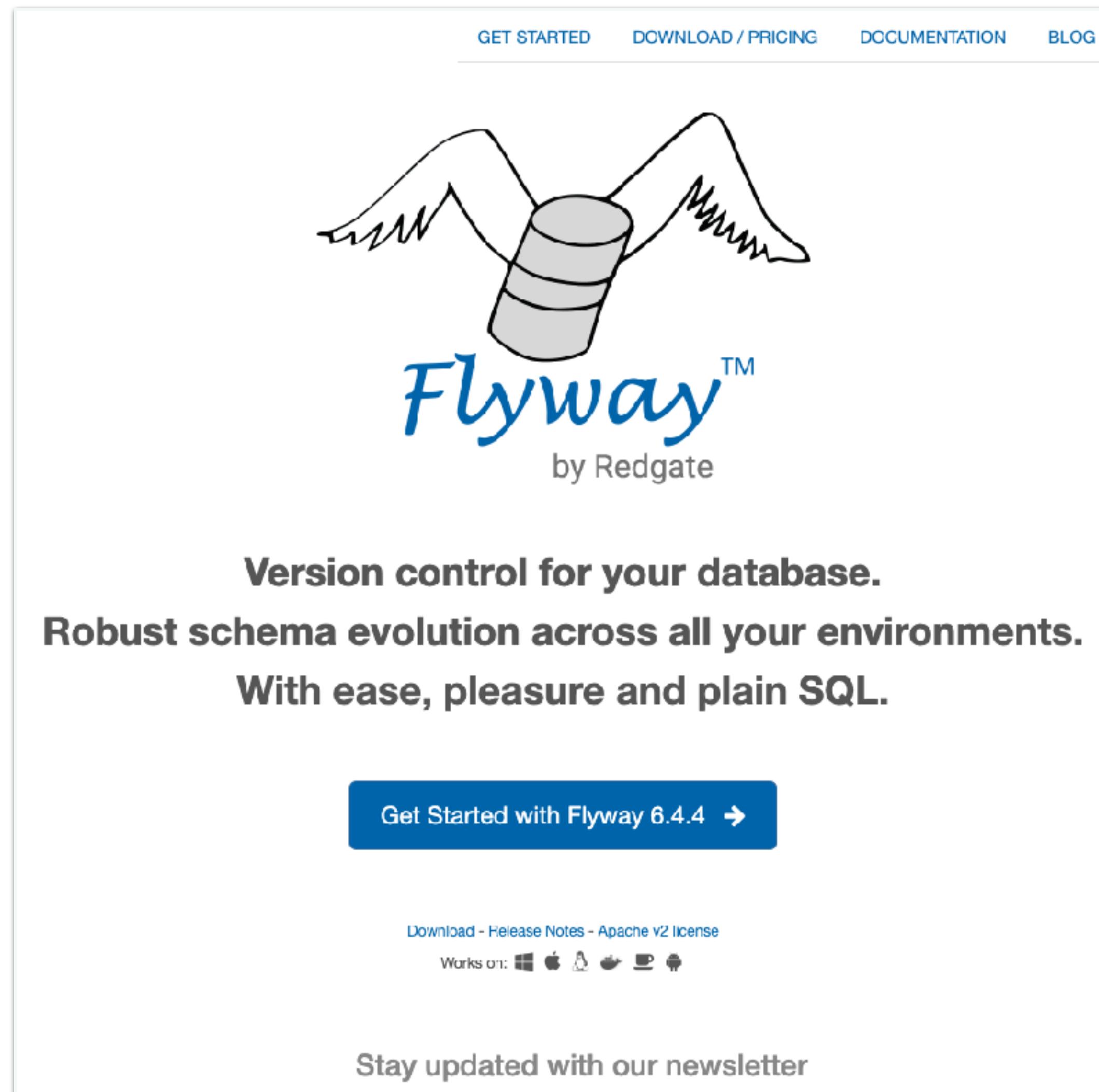
Each DB refactoring is stored in version control

Can be run in a safe, deterministic, idempotent manner

<https://flywaydb.org/>

@samnewman

## DATA REFACTORING TOOLS



The screenshot shows the homepage of the Flyway website. At the top, there is a navigation bar with links for "GET STARTED", "DOWNLOAD / PRICING", "DOCUMENTATION", and "BLOG". Below the navigation bar is the Flyway logo, which features a stylized grey cylinder (resembling a database) positioned between two wavy lines that suggest mountains or waves. The word "Flyway" is written in a blue, lowercase, sans-serif font next to the cylinder, with a trademark symbol (TM) at the end. Below the logo, the text "by Redgate" is visible. The main headline on the page reads "Version control for your database. Robust schema evolution across all your environments. With ease, pleasure and plain SQL." Below this headline is a blue button with white text that says "Get Started with Flyway 6.4.4 →". At the bottom of the page, there are links for "Download - Release Notes - Apache v2 license" and "Works on: Windows, macOS, Linux, Oracle Database, PostgreSQL, MySQL, Microsoft SQL Server, IBM DB2, SAP HANA, SQLite, Oracle Database, MySQL, PostgreSQL, Microsoft SQL Server, IBM DB2, SAP HANA, SQLite". There is also a section for staying updated with the newsletter.

Each DB refactoring is stored in version control

Can be run in a safe, deterministic, idempotent manner

Do it throughout your deployment pipeline to ensure it's rigorously tested

<https://flywaydb.org/>

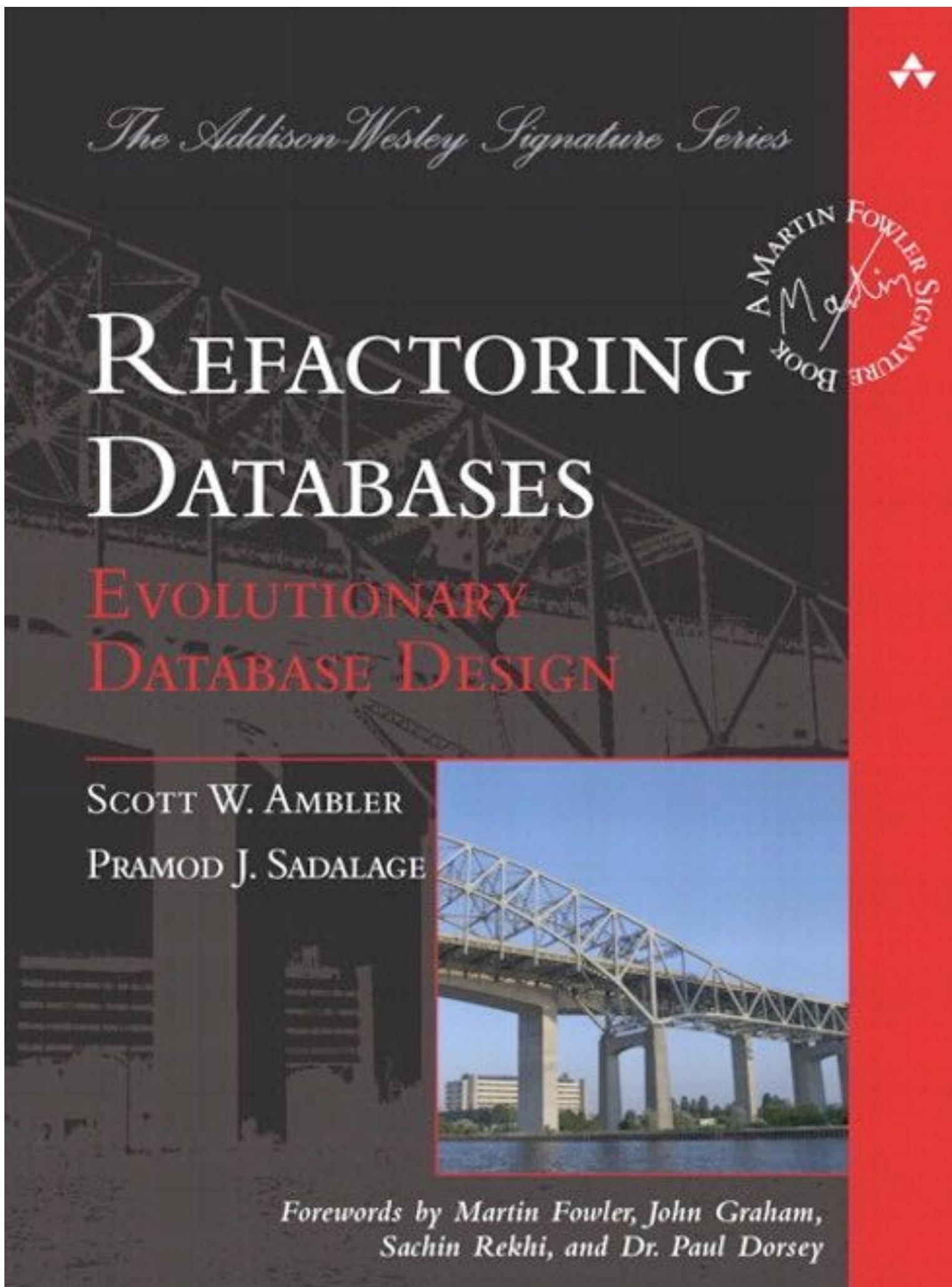
@samnewman

## POLL: HAVE YOU USED A DB REFACTORING TOOL?

Yes

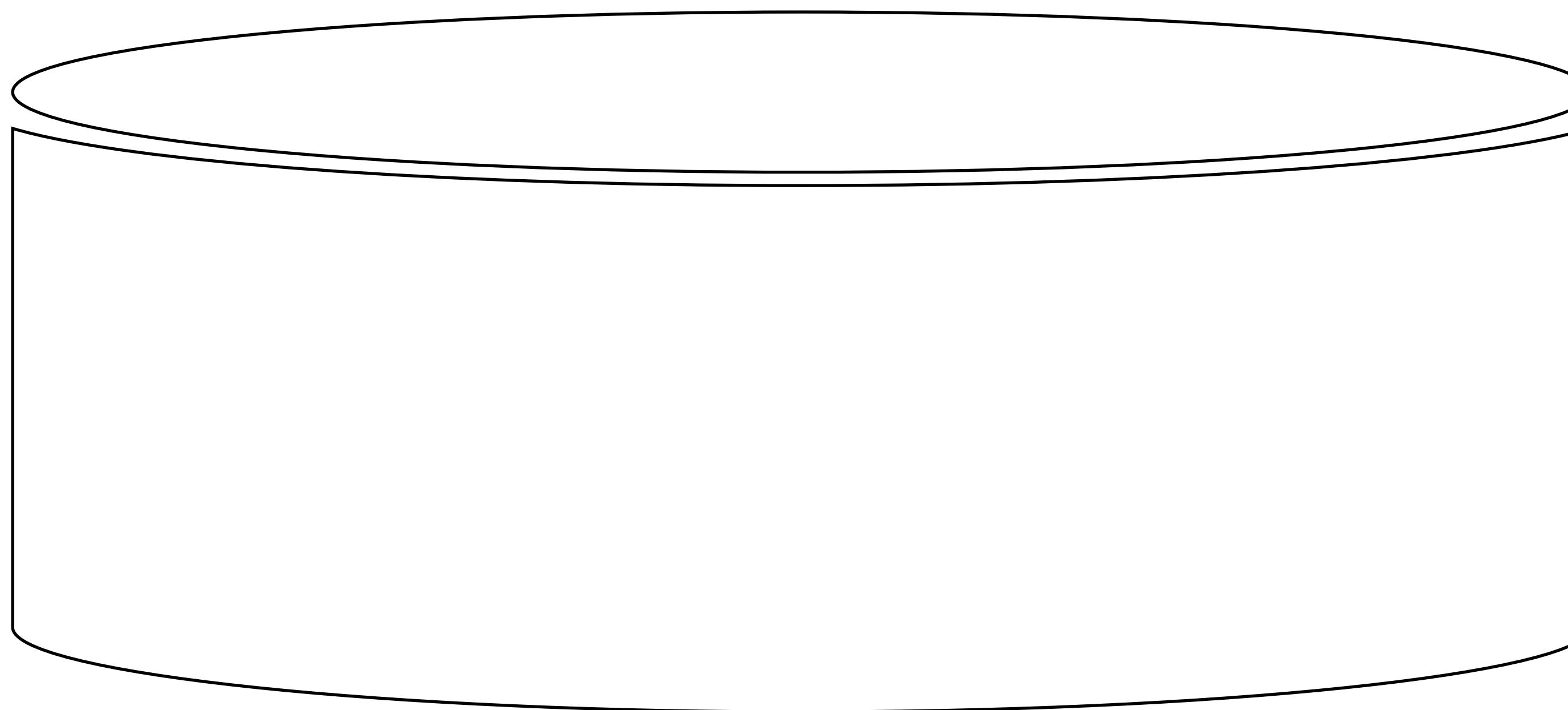
No

## REFACTORING DATABASES - FURTHER READING

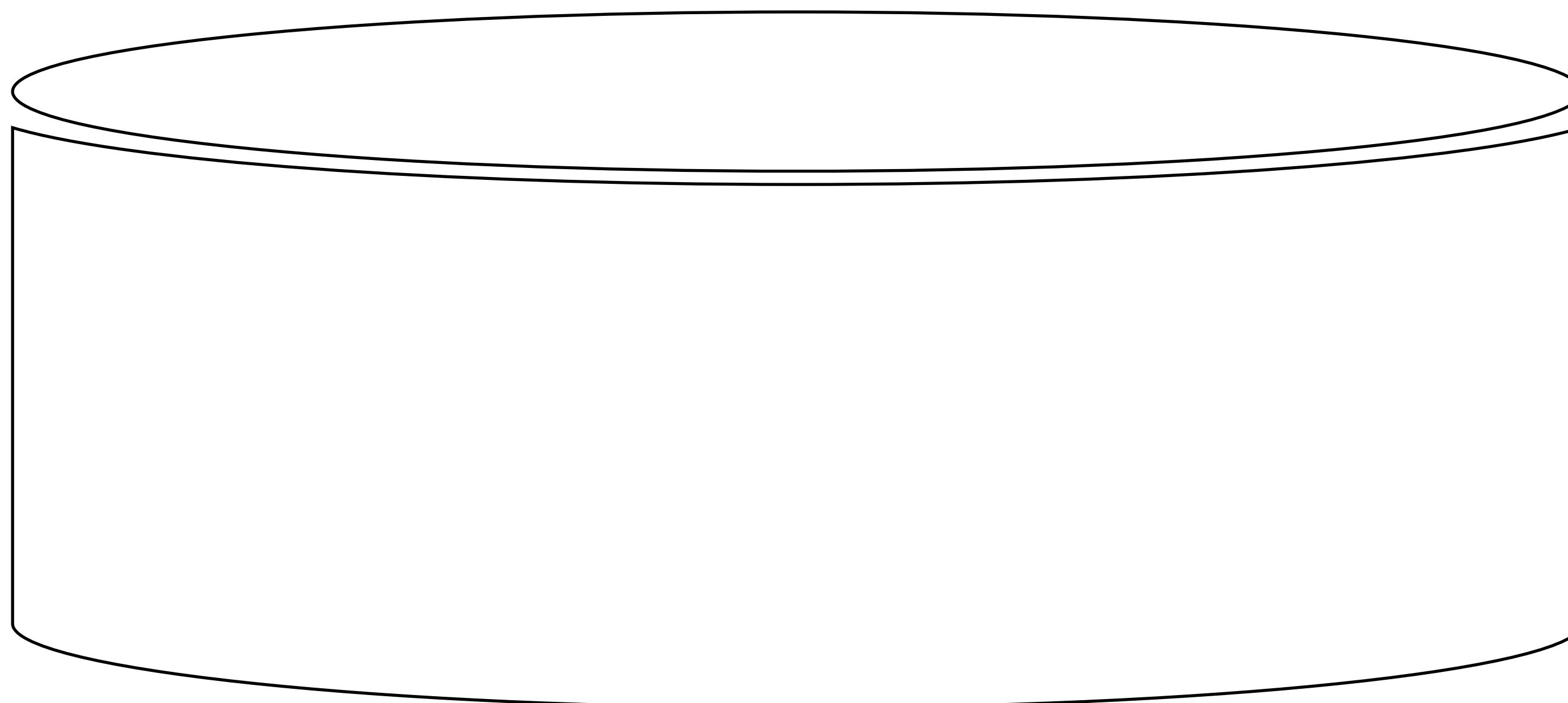
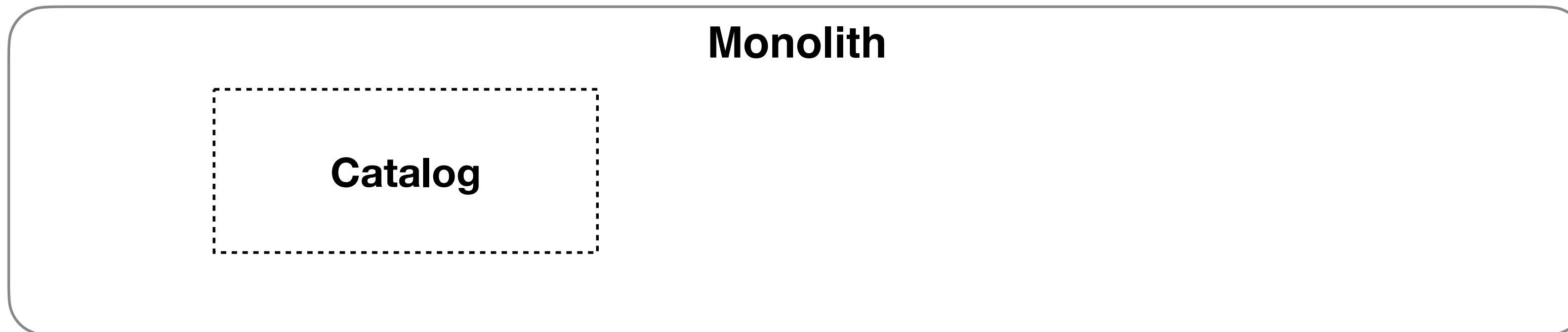


**TOP CHARTS!**

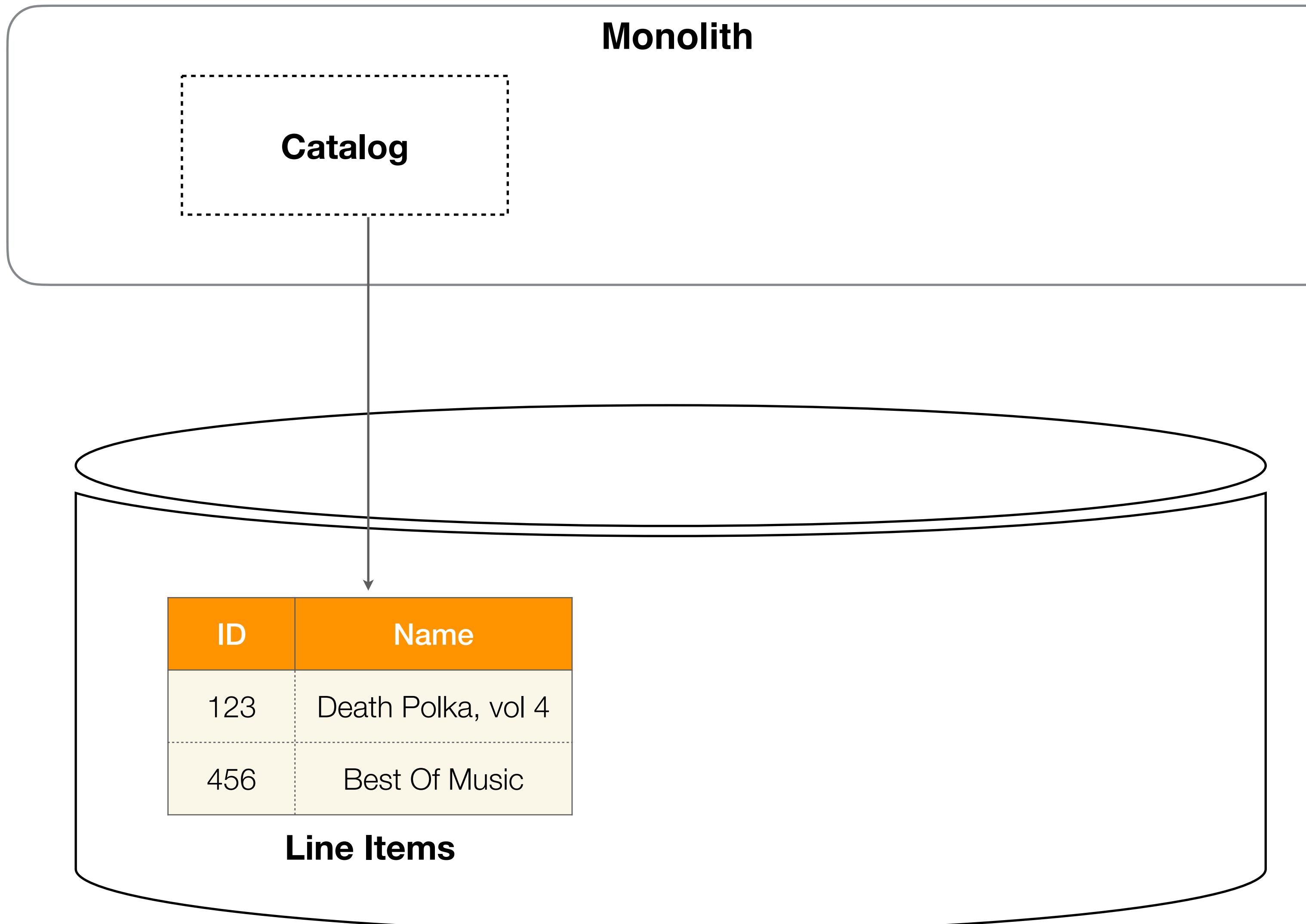
**Monolith**



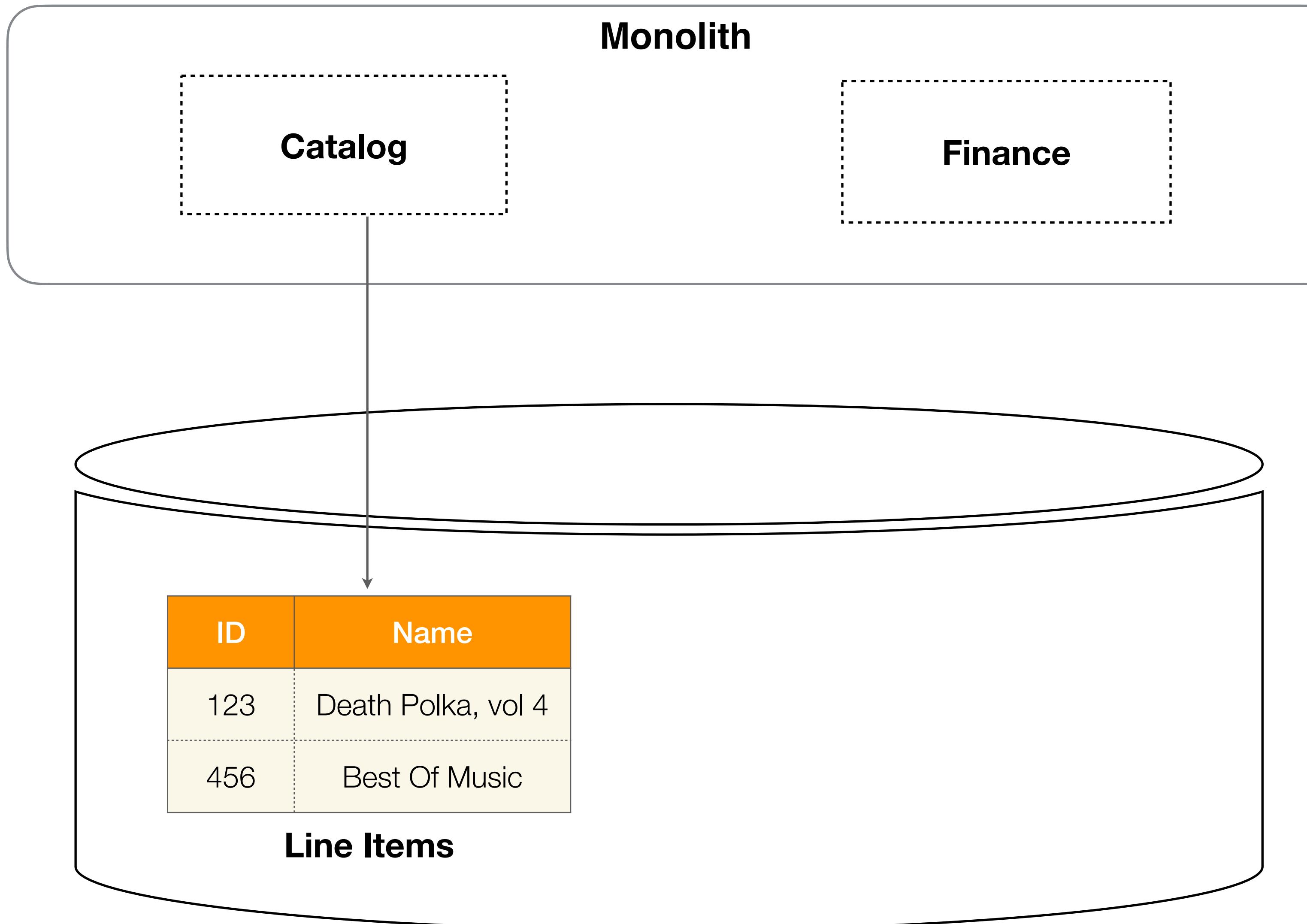
# TOP CHARTS!



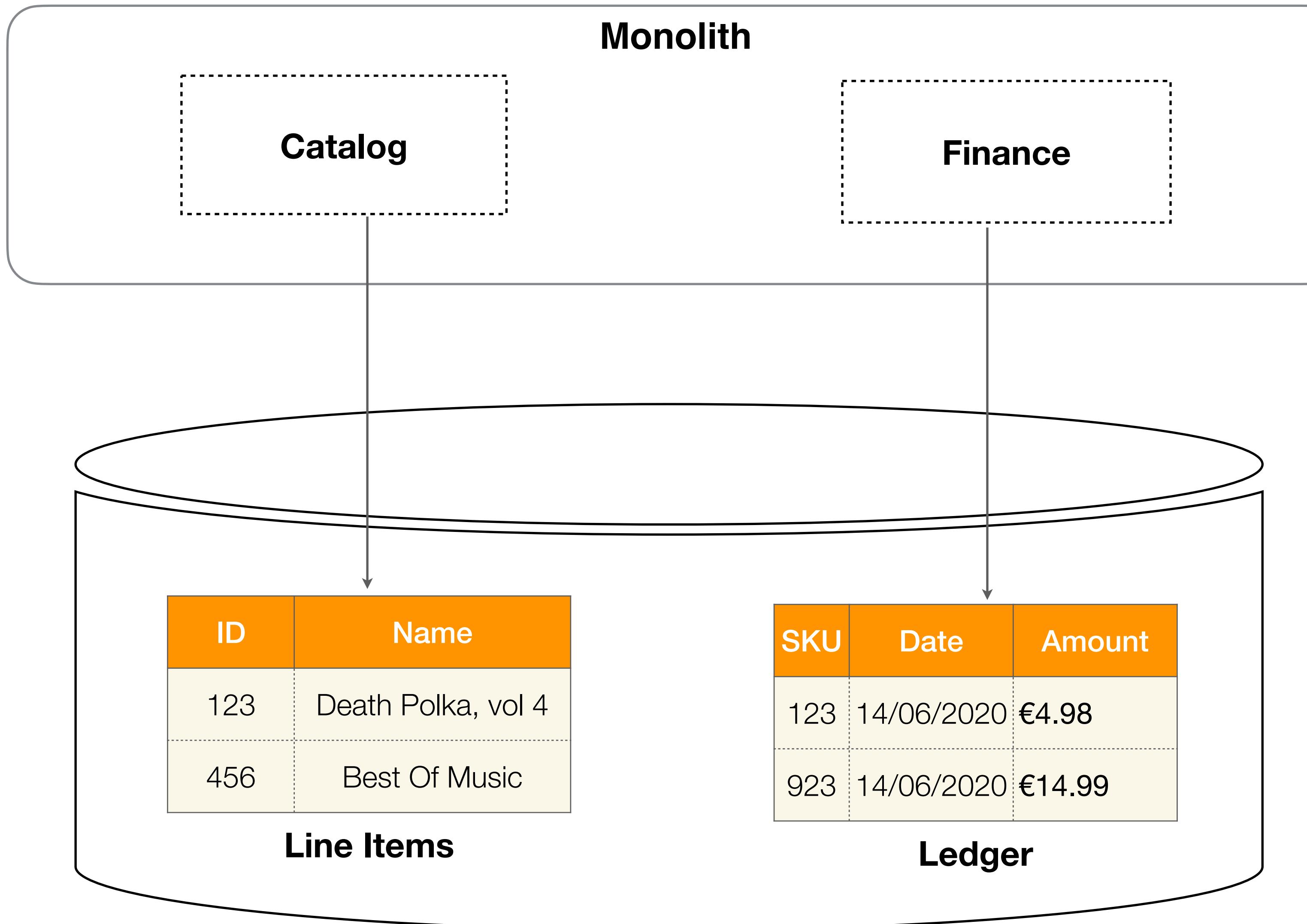
# TOP CHARTS!



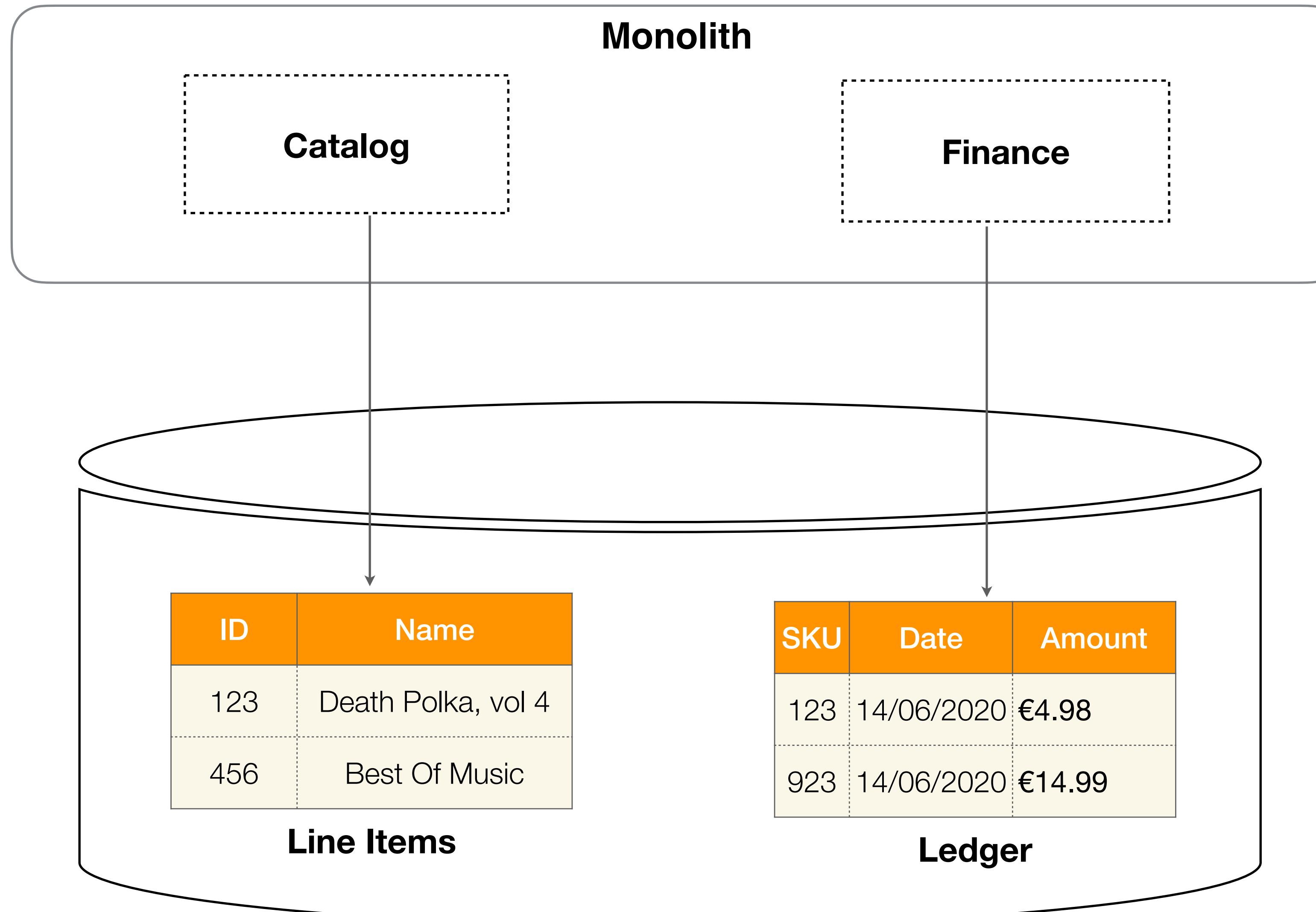
# TOP CHARTS!



# TOP CHARTS!



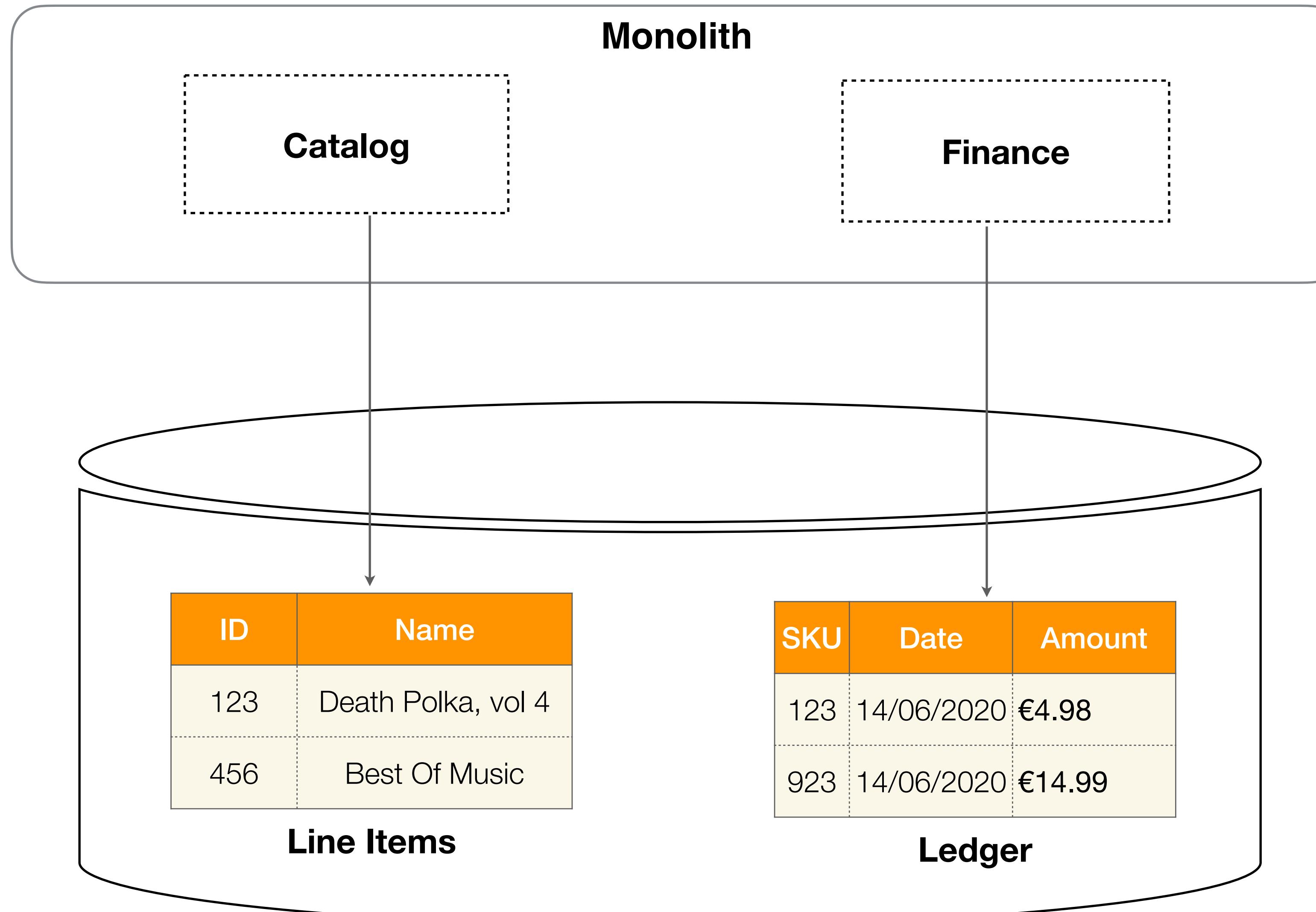
# TOP CHARTS!



## Best Sellers This Week!

1	Death Polka, Vol 4
2	Greatest Hits of Peter Andre
3	Now That's What I Call Hoovercore

# TOP CHARTS!

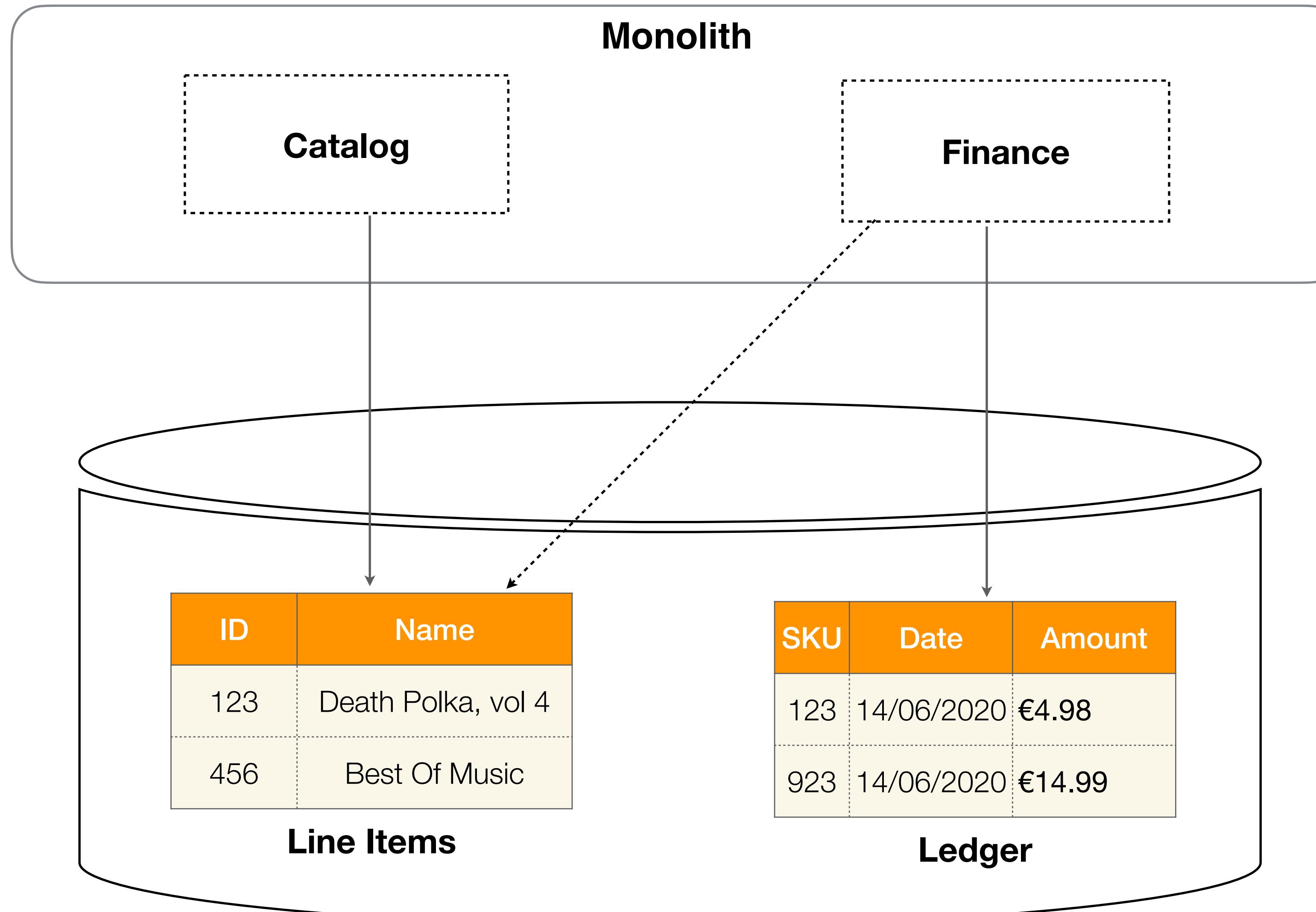


## Best Sellers This Week!

1	Death Polka, Vol 4
2	Greatest Hits of Peter Andre
3	Now That's What I Call Hoovercore

The ledger contains the ID of what sold when, but we need the name....

# TOP CHARTS!

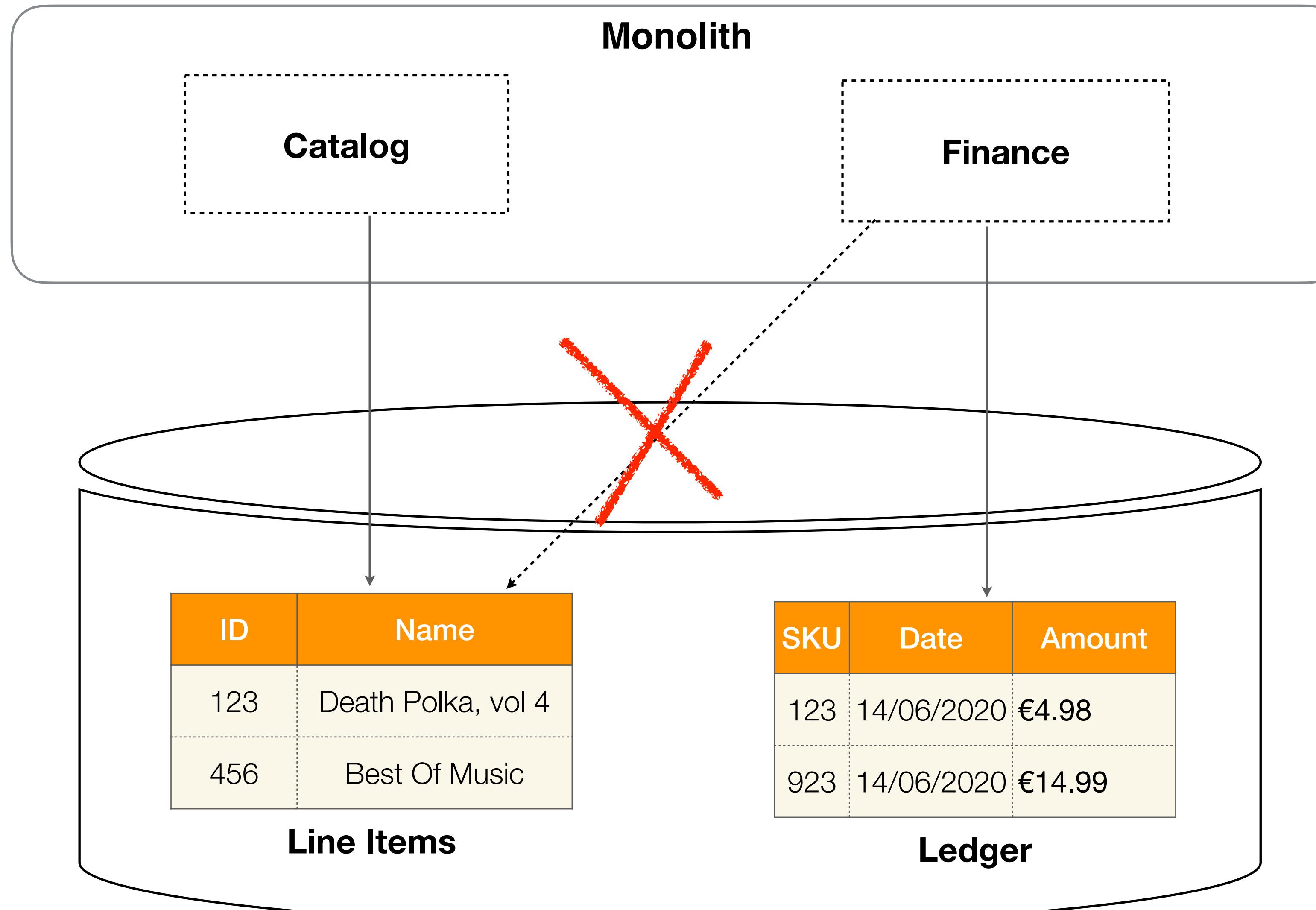


## Best Sellers This Week!

1	Death Polka, Vol 4
2	Greatest Hits of Peter Andre
3	Now That's What I Call Hoovercore

**The ledger contains the ID of what sold when, but we need the name....**

# TOP CHARTS!

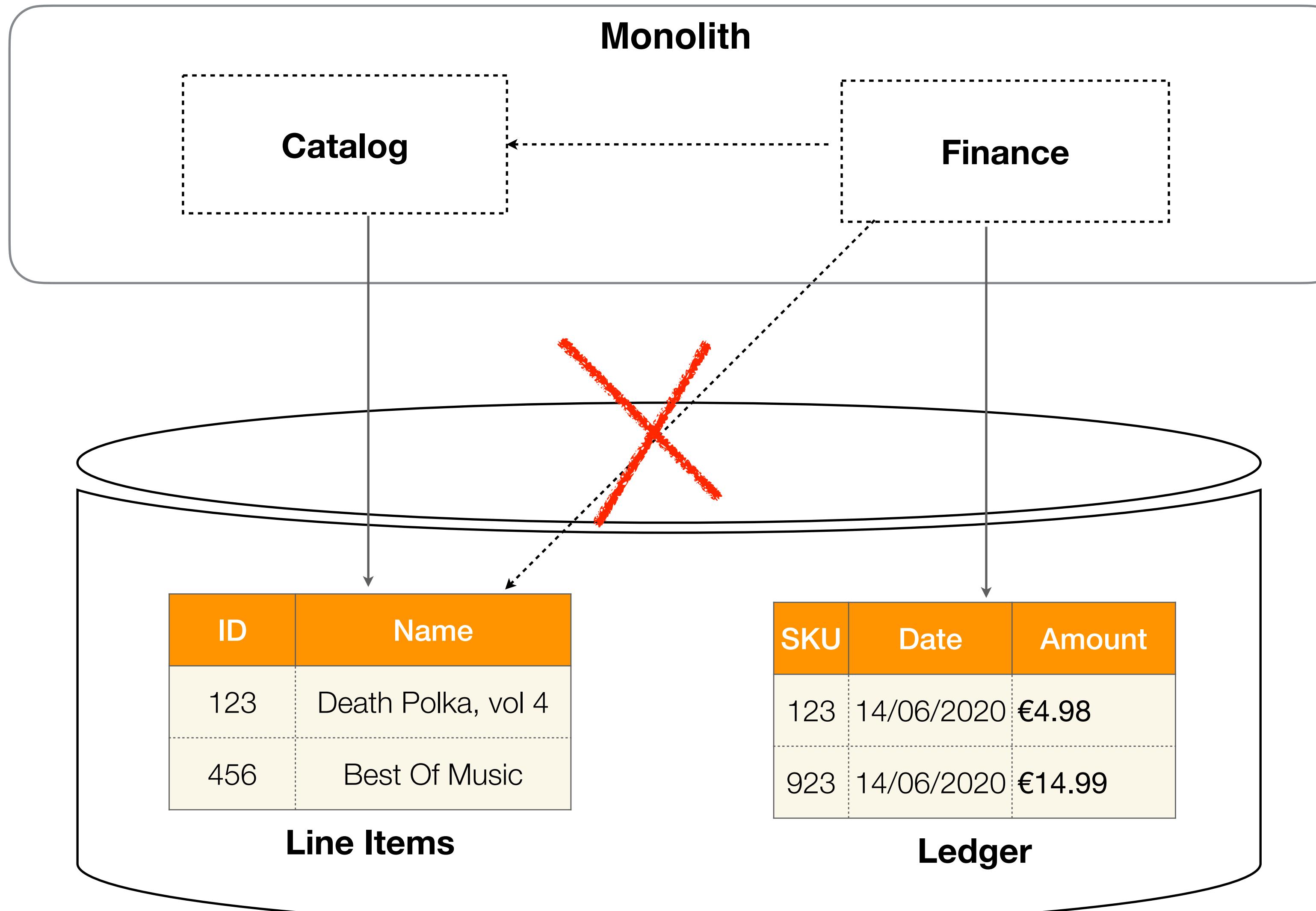


## Best Sellers This Week!

1	Death Polka, Vol 4
2	Greatest Hits of Peter Andre
3	Now That's What I Call Hoovercore

**The ledger contains the ID of what sold when, but we need the name....**

# TOP CHARTS!

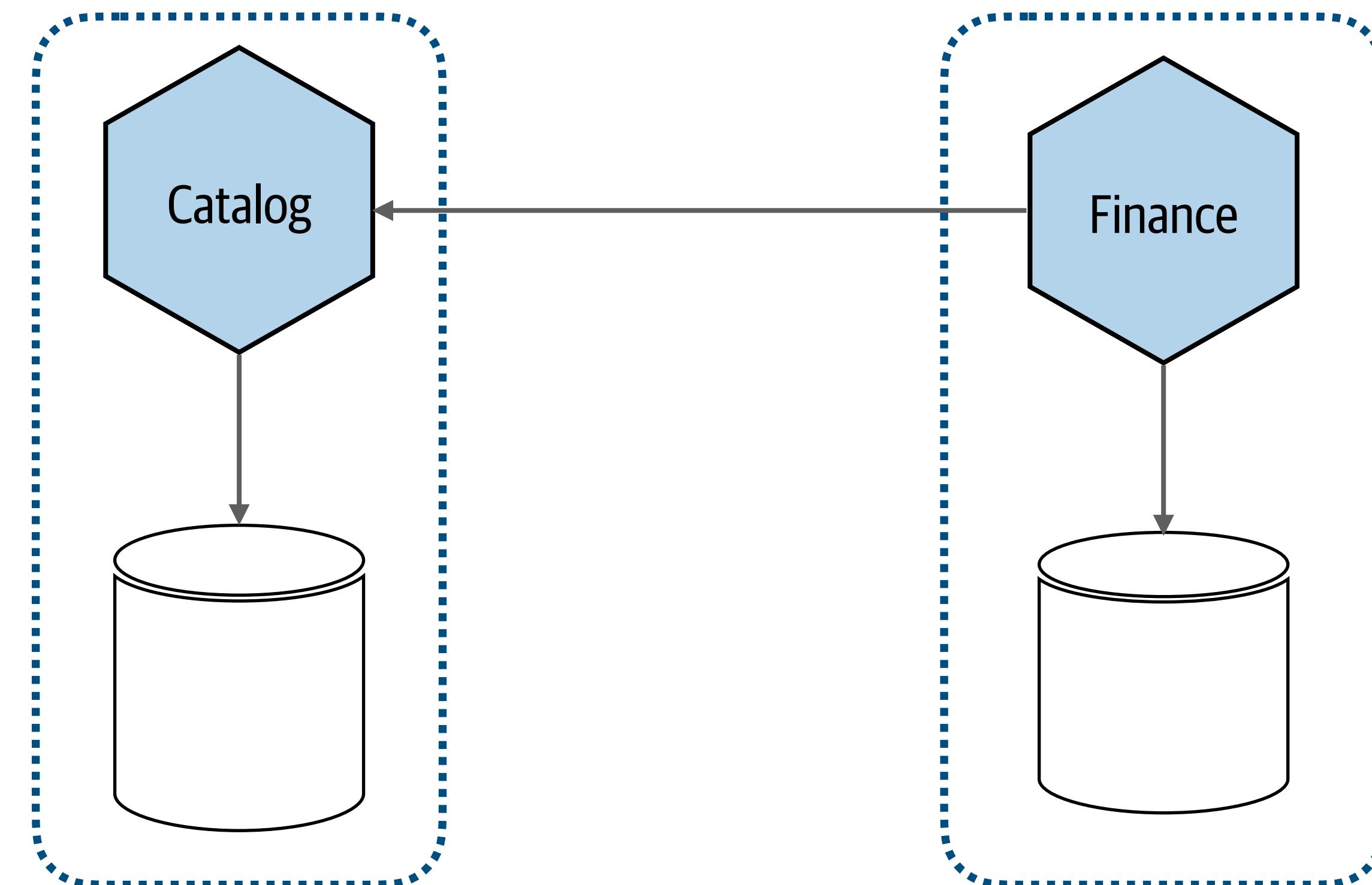


## Best Sellers This Week!

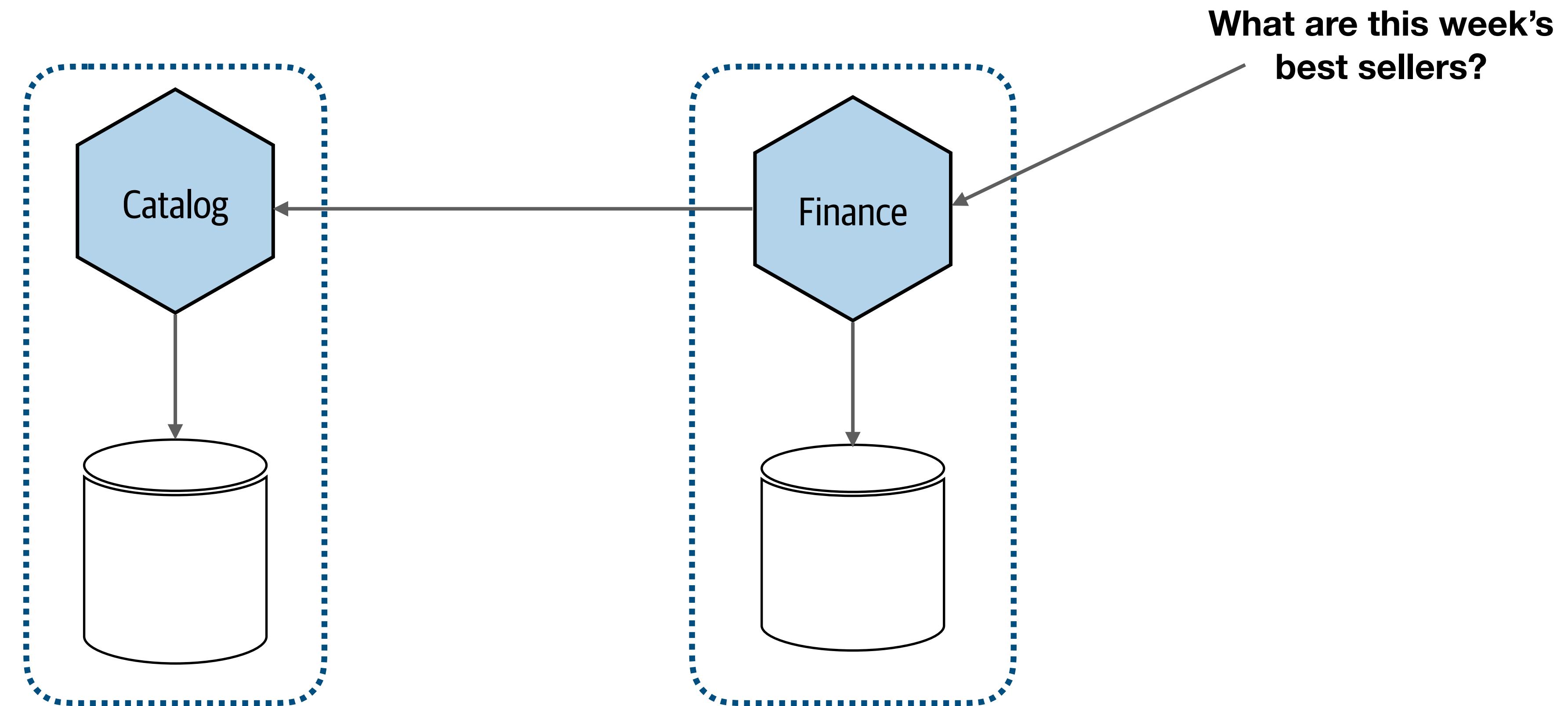
1	Death Polka, Vol 4
2	Greatest Hits of Peter Andre
3	Now That's What I Call Hoovercore

**The ledger contains the ID of what sold when, but we need the name....**

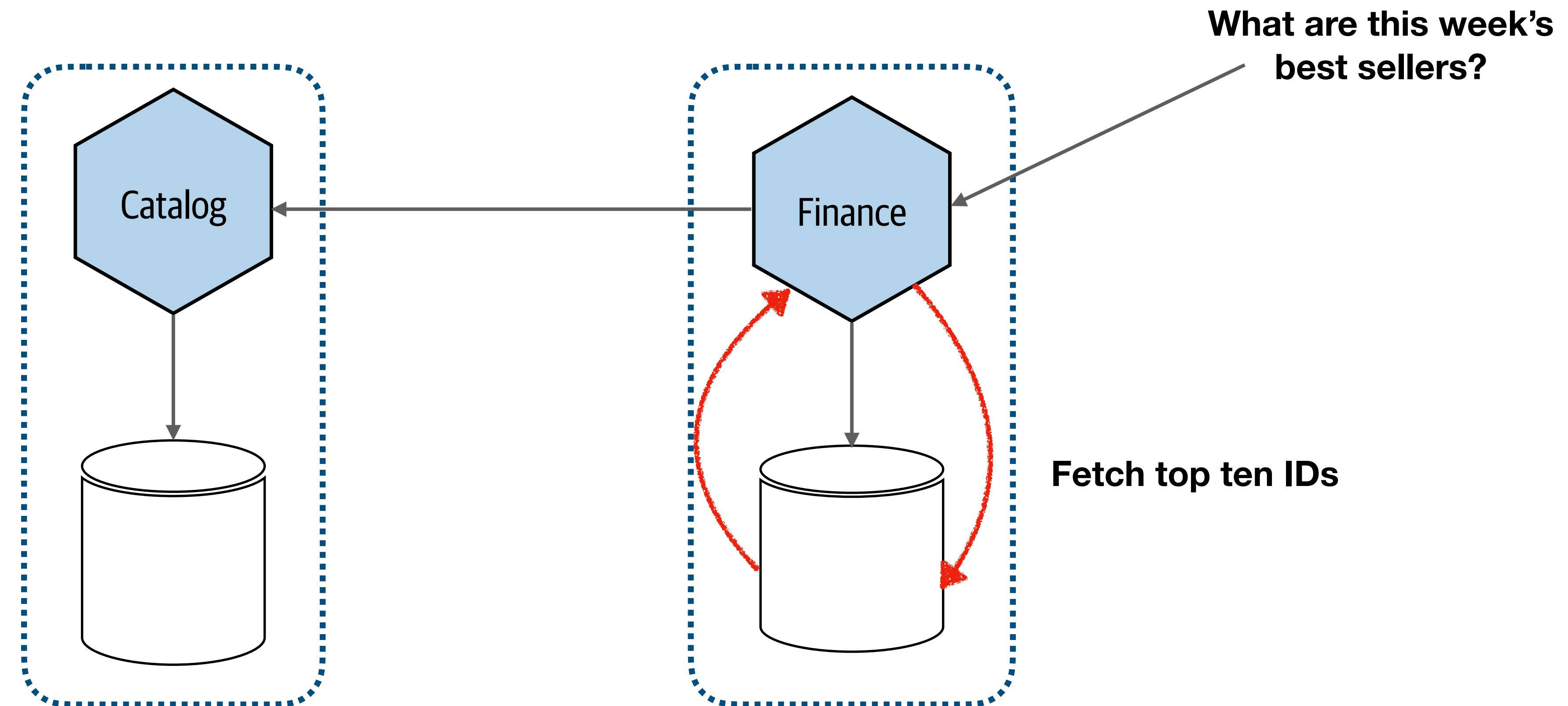
## JOINS AT THE SERVICES TIER



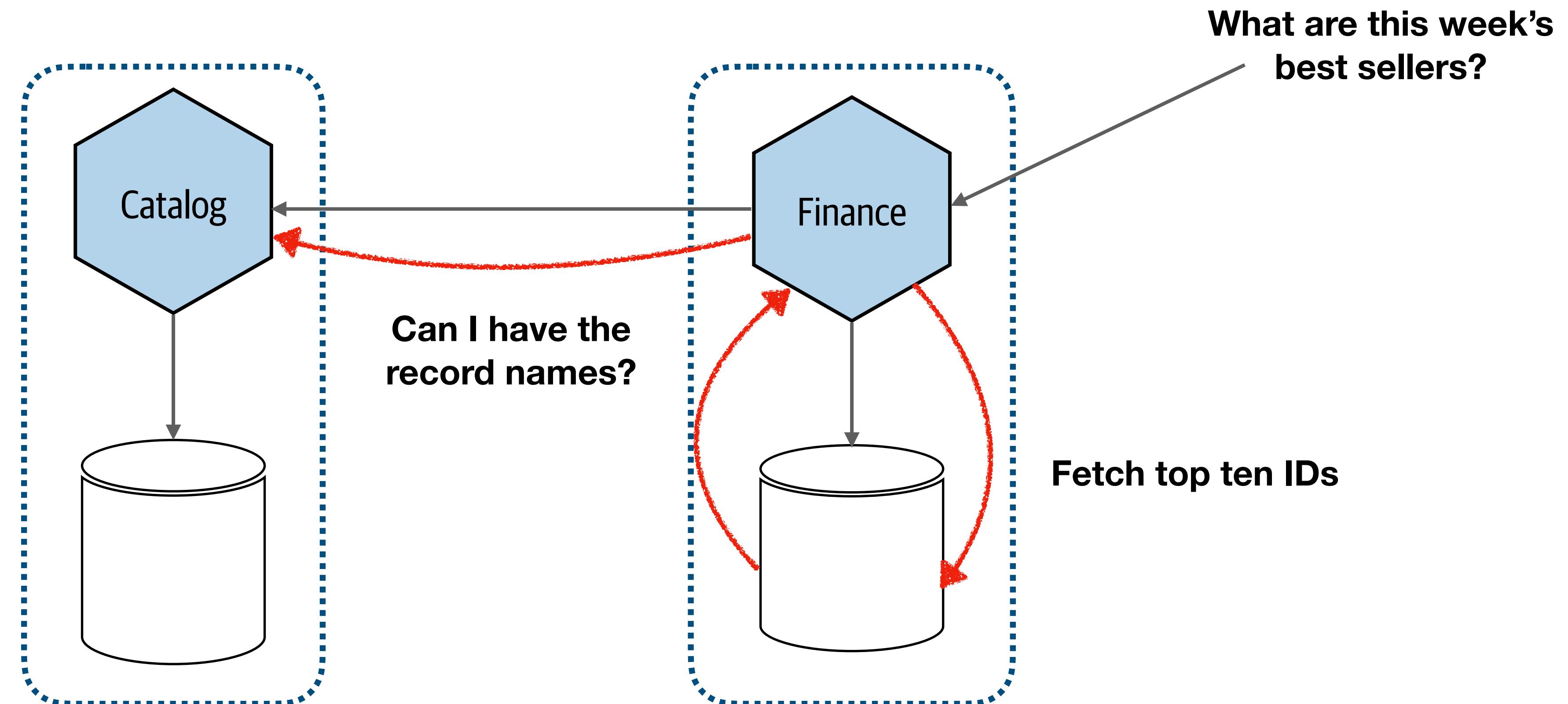
## JOINS AT THE SERVICES TIER



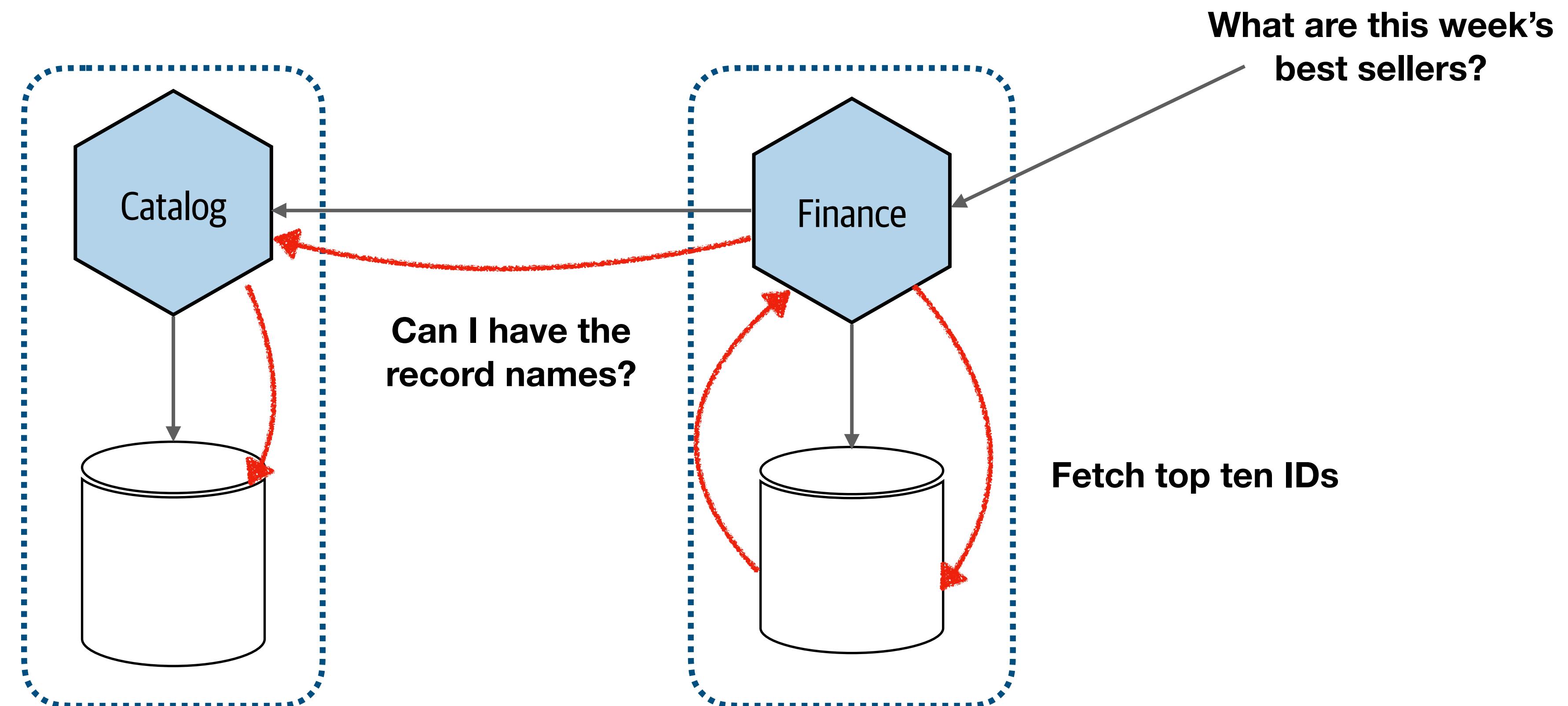
## JOINS AT THE SERVICES TIER



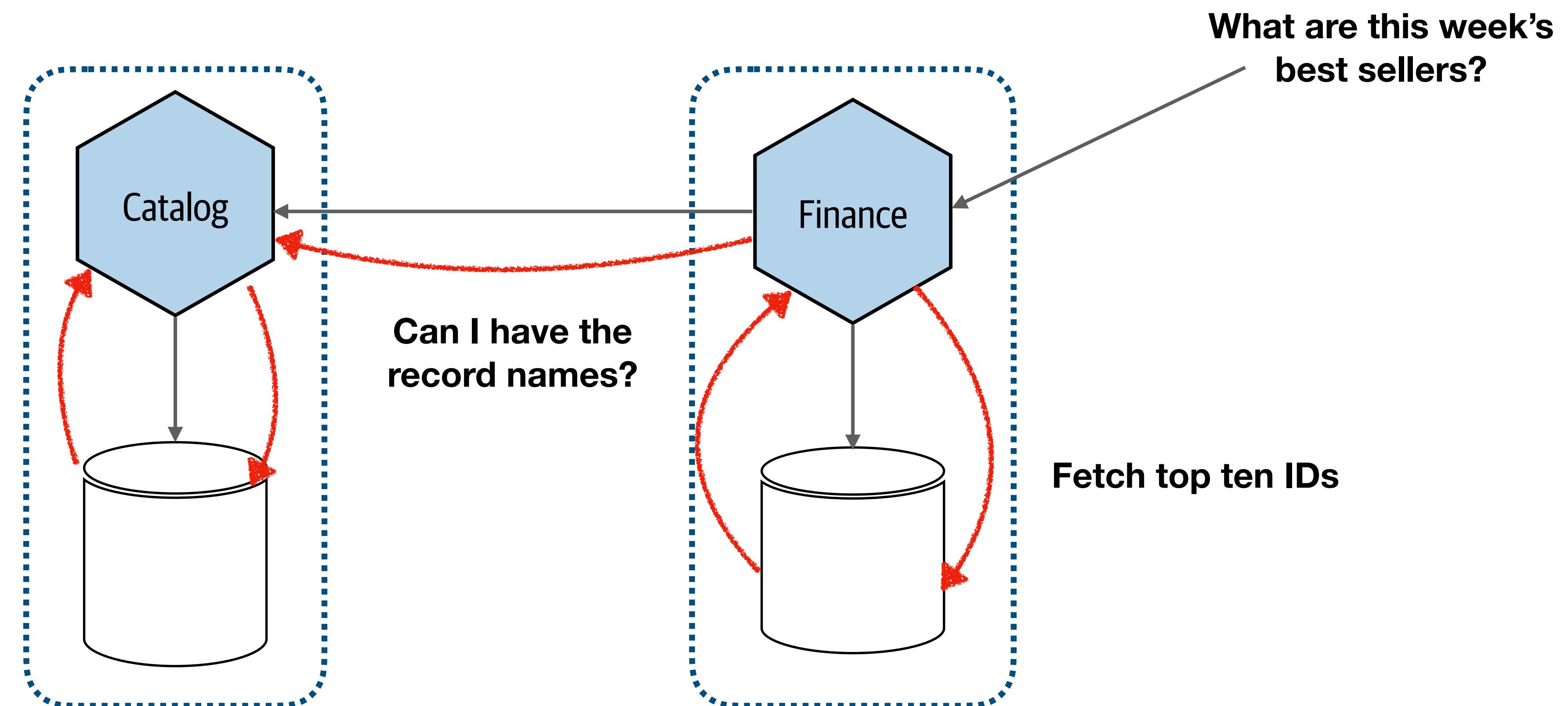
## JOINS AT THE SERVICES TIER



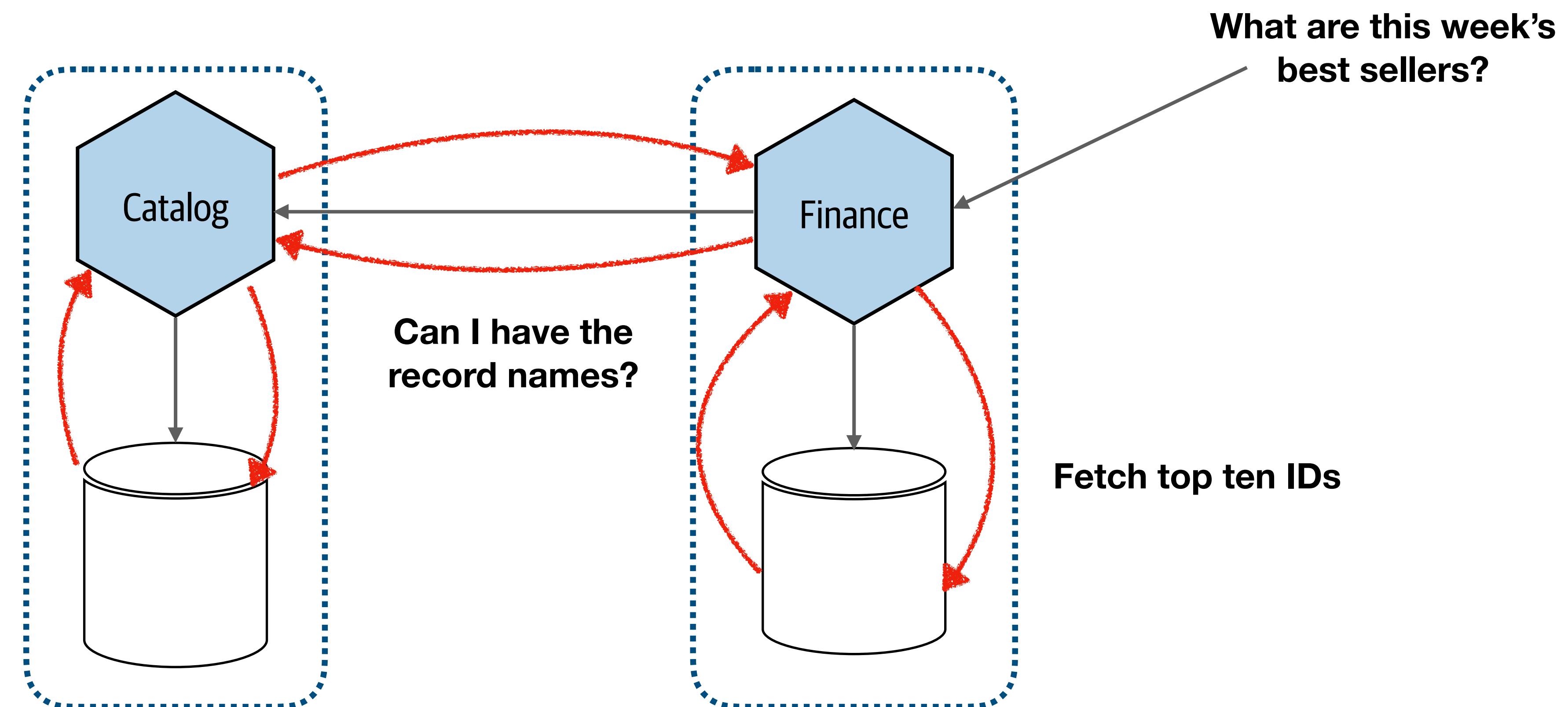
## JOINS AT THE SERVICES TIER



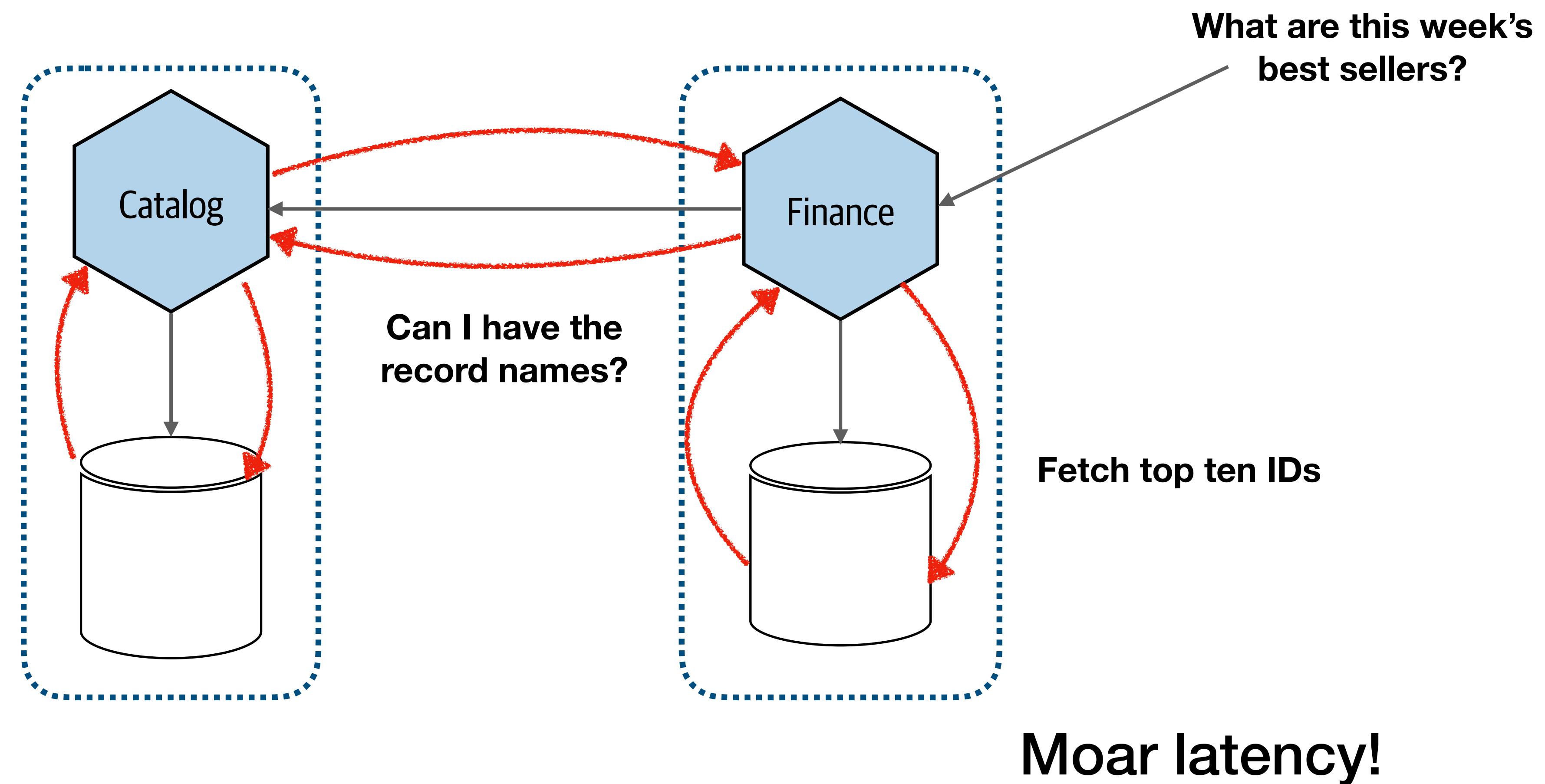
## JOINS AT THE SERVICES TIER



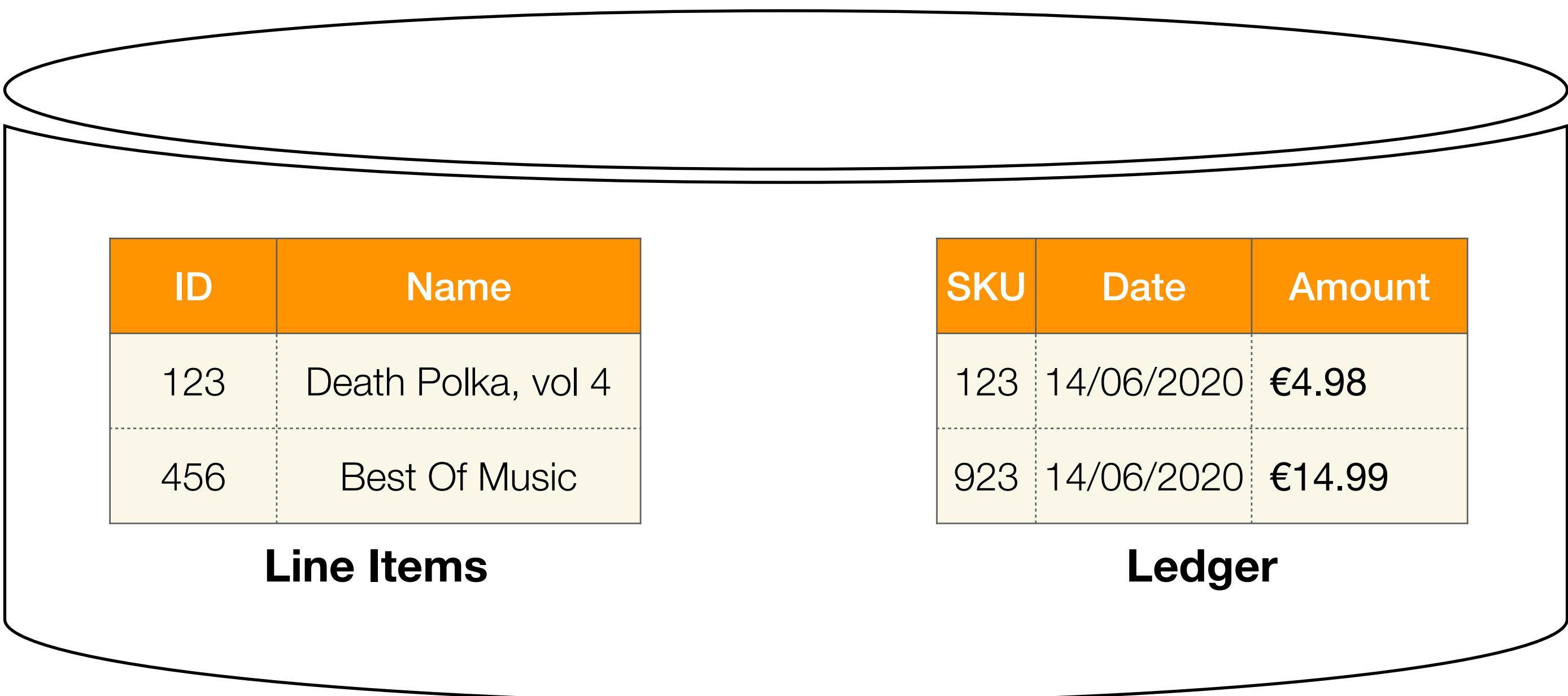
## JOINS AT THE SERVICES TIER



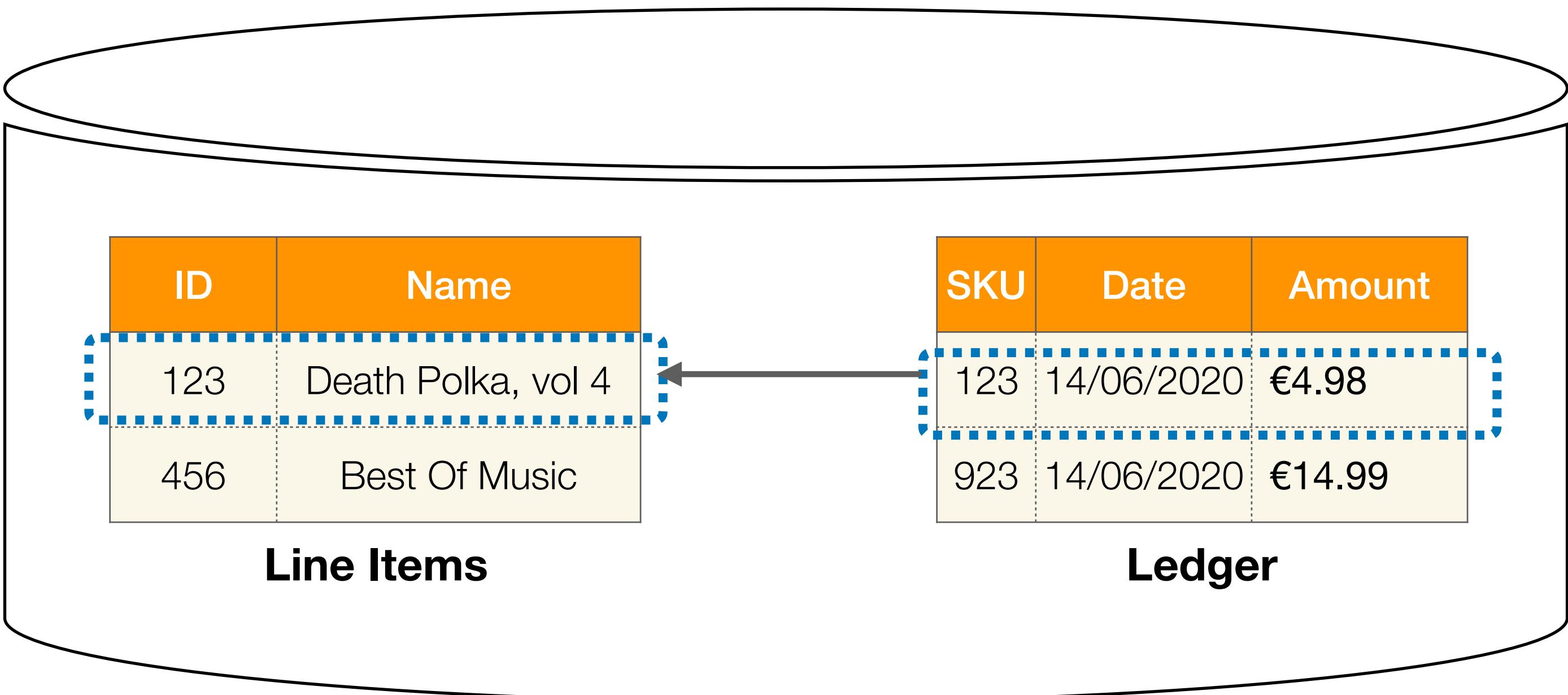
## JOINS AT THE SERVICES TIER



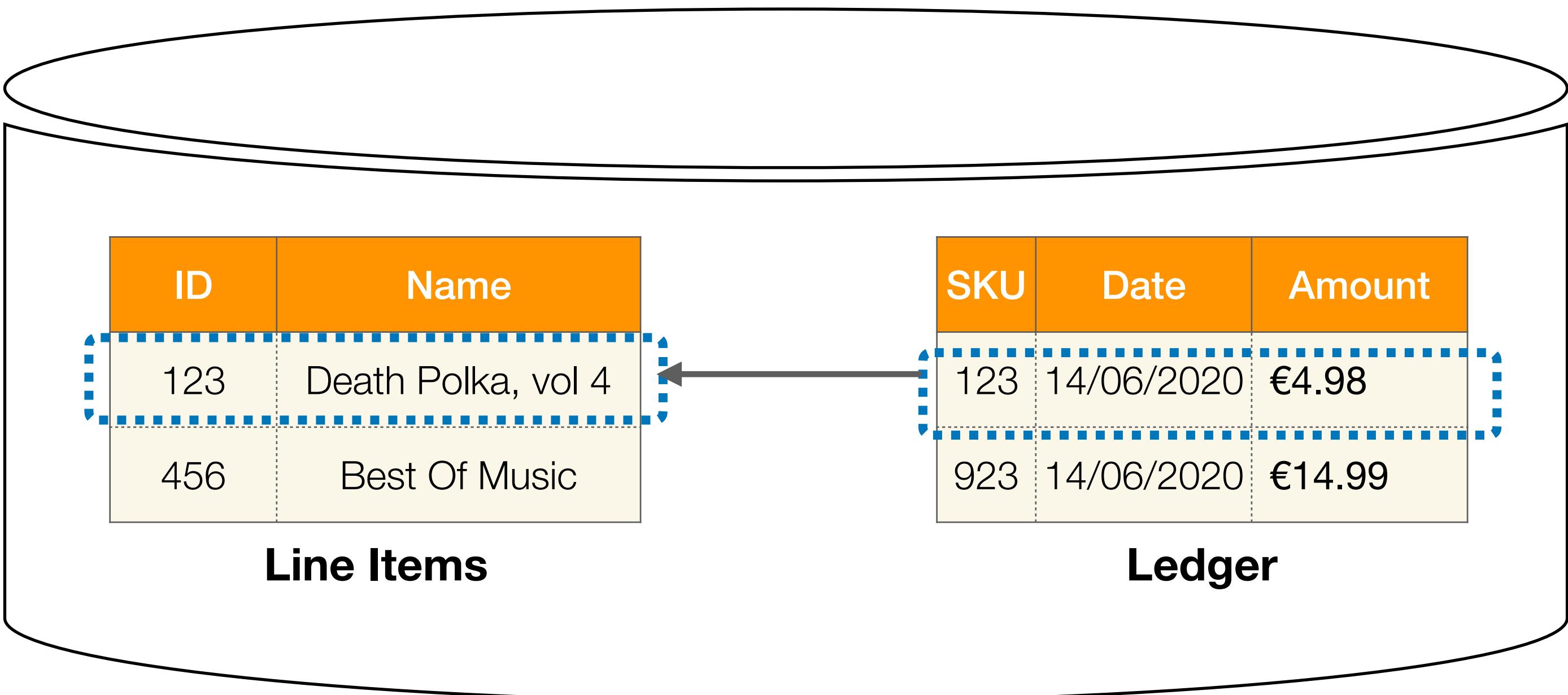
# FOREIGN KEY RELATIONSHIPS



# FOREIGN KEY RELATIONSHIPS

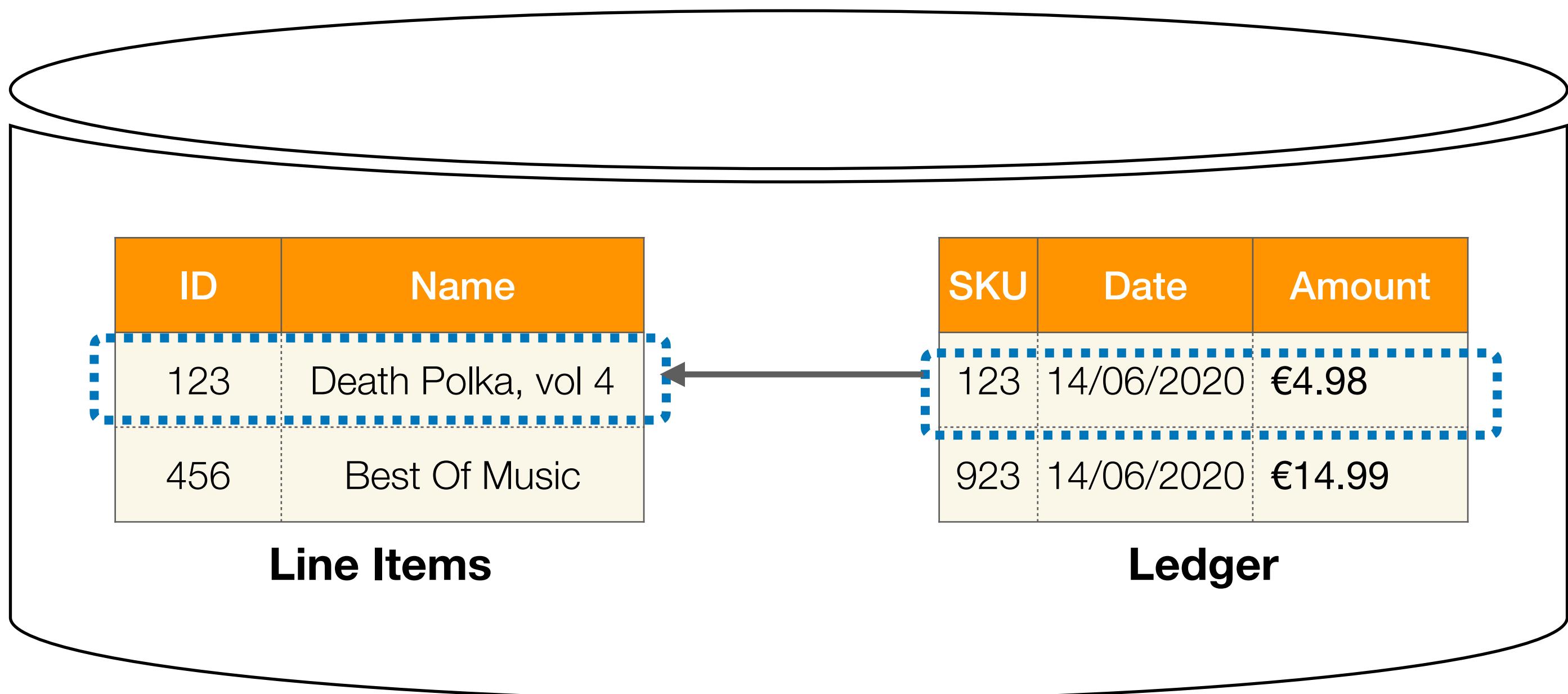


## FOREIGN KEY RELATIONSHIPS



Enforce referential integrity

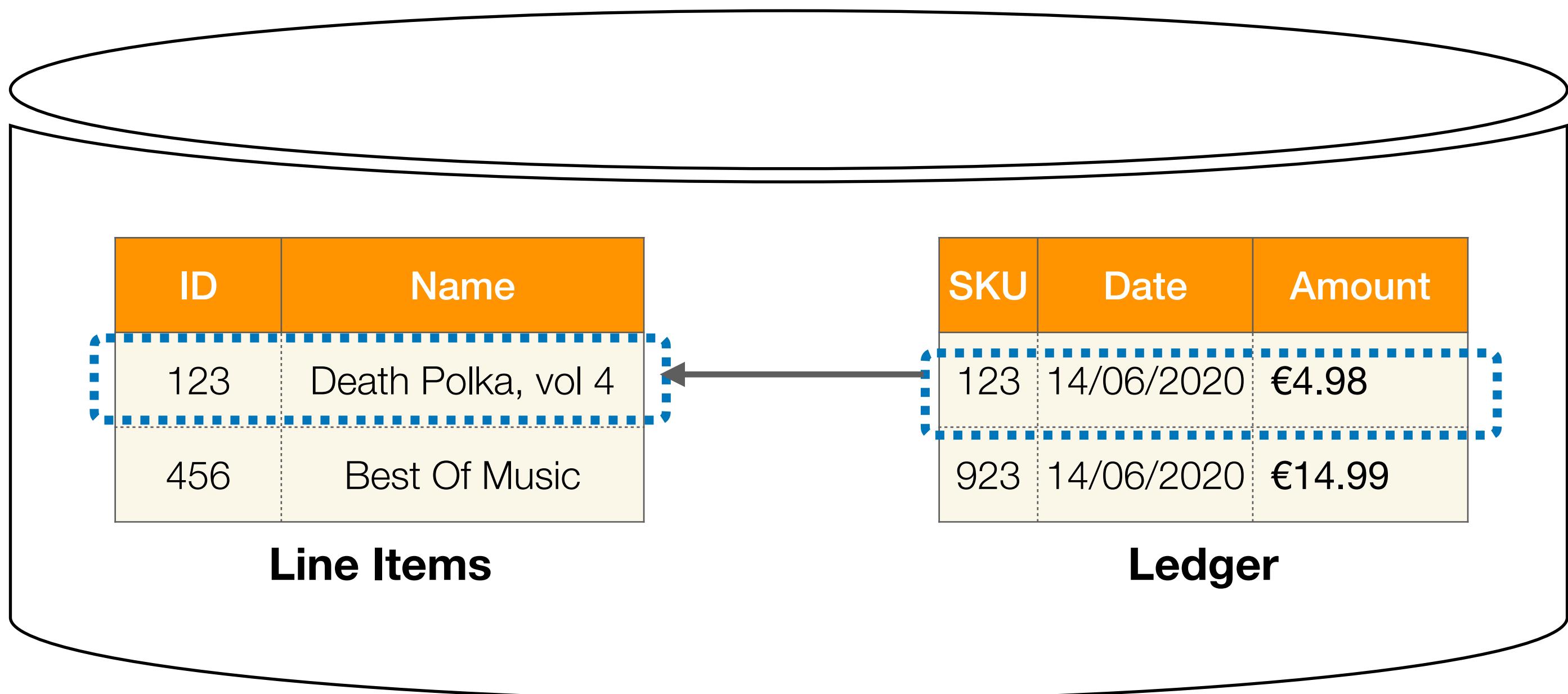
## FOREIGN KEY RELATIONSHIPS



Enforce referential integrity

Make relationships explicit

## FOREIGN KEY RELATIONSHIPS



**Enforce referential integrity**

**Make relationships explicit**

**Aid join performance**

# MAKING REFERENCES EXPLICIT

## Pattern: Using Pseudo-URIs with Microservices

Mar 22, 2017

• Microservices • Distributed Systems • Patterns •

I've spent some time talking about [the very basics you need to have in place before thinking about going down a microservices route](#), but even if you have these in place that doesn't mean that you aren't going to find some new surprises. Microservices impose a very distributed architecture, and this requires us to re-visit many concepts that are considered a solved problem in more traditional scenarios.

One of such concepts is how we implement identities for objects. This is usually a no-brainer in more monolithic architectures but it becomes a more interesting challenge as we have more distribution and collaboration between services.

### How I understand object identity

Before we discuss how things change in a distributed services scenario, let's try to build a working definition of object identity.

The first thing to clarify is what mean by the word *object* in this text. While we are going to use some

## pURIs: Borrowing a solution from the Internet

When facing the problems above, my team at SoundCloud started exploring alternatives that would allow for us to have simple, scalar values that were still rich enough to act as good identifiers across our hundreds of microservices. Reading through decades of industry work on the matter, we found something simple that could help us: [Uniform Resource Names, or URNs](#). URNs were a type of [Uniform Resource Identifiers \(URIs\)](#) that, as opposed to [URLs](#), were only concerned with the *identifier* for a resource, but not with how to locate it.

The specification allowed for us to create structured identifiers that would contain information about the object's type. This means that instead of the confusion created by an `id` of `123`, we would have `id`s that looked like:

```
urn:tracks:123  
urn:users:123  
urn:comments:123  
urn:artwork:123  
urn:playlists:123
```

[https://philcalcado.com/2017/03/22/pattern\\_using\\_seudo-uris\\_with\\_microservices.html](https://philcalcado.com/2017/03/22/pattern_using_seudo-uris_with_microservices.html)

# MAKING REFERENCES EXPLICIT

## Pattern: Using Pseudo-URIs with Microservices

## pURIs: Borrowing a solution from the Internet

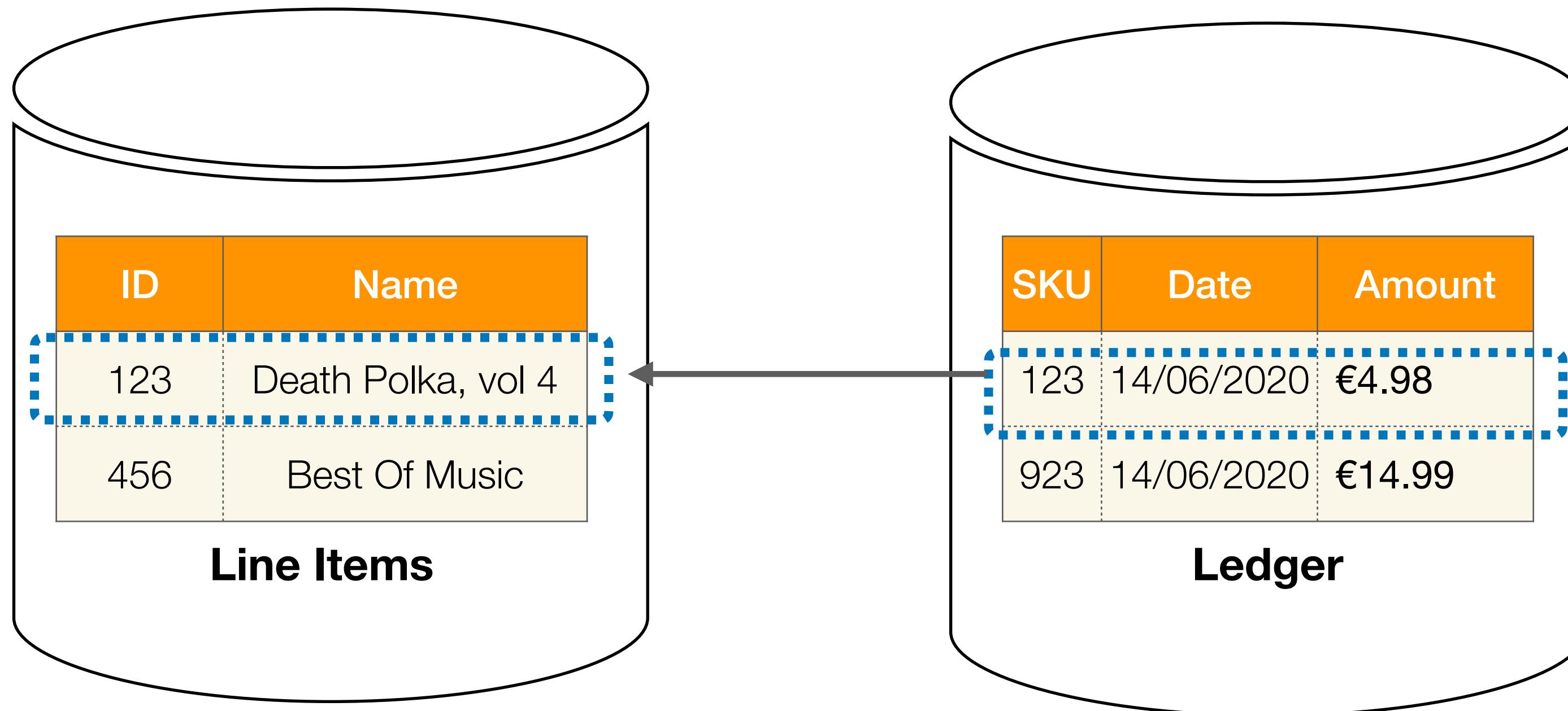
When facing the problems above, my team at SoundCloud started exploring alternatives that would

```
soundcloud:tracks:123  
soundcloud:users:123  
soundcloud:comments:123  
soundcloud:artwork:123  
soundcloud:playlists:123
```

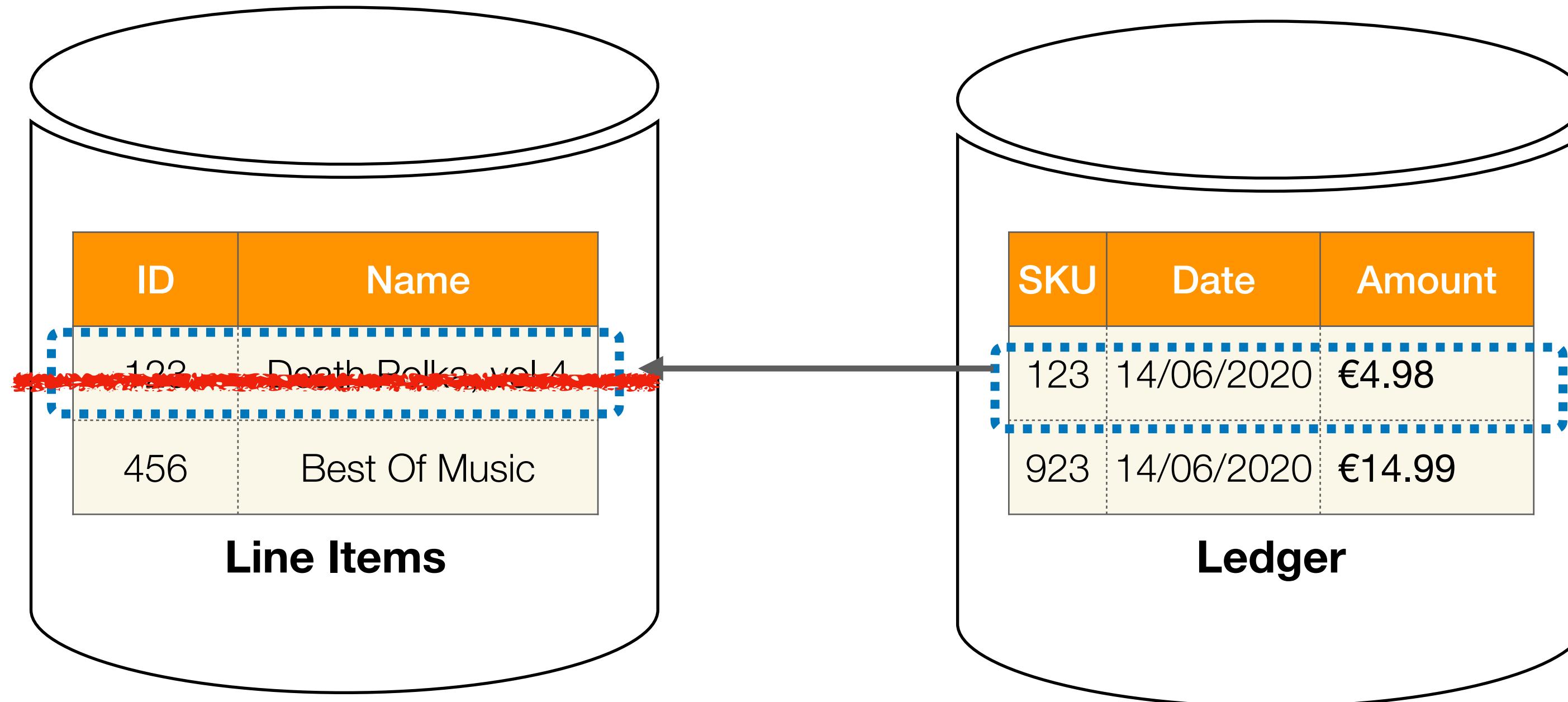
The first thing to clarify is what mean by the word *object* in this text. While we are going to use some

[https://philcalcado.com/2017/03/22/pattern\\_using\\_seudo-uris\\_with\\_microservices.html](https://philcalcado.com/2017/03/22/pattern_using_seudo-uris_with_microservices.html)

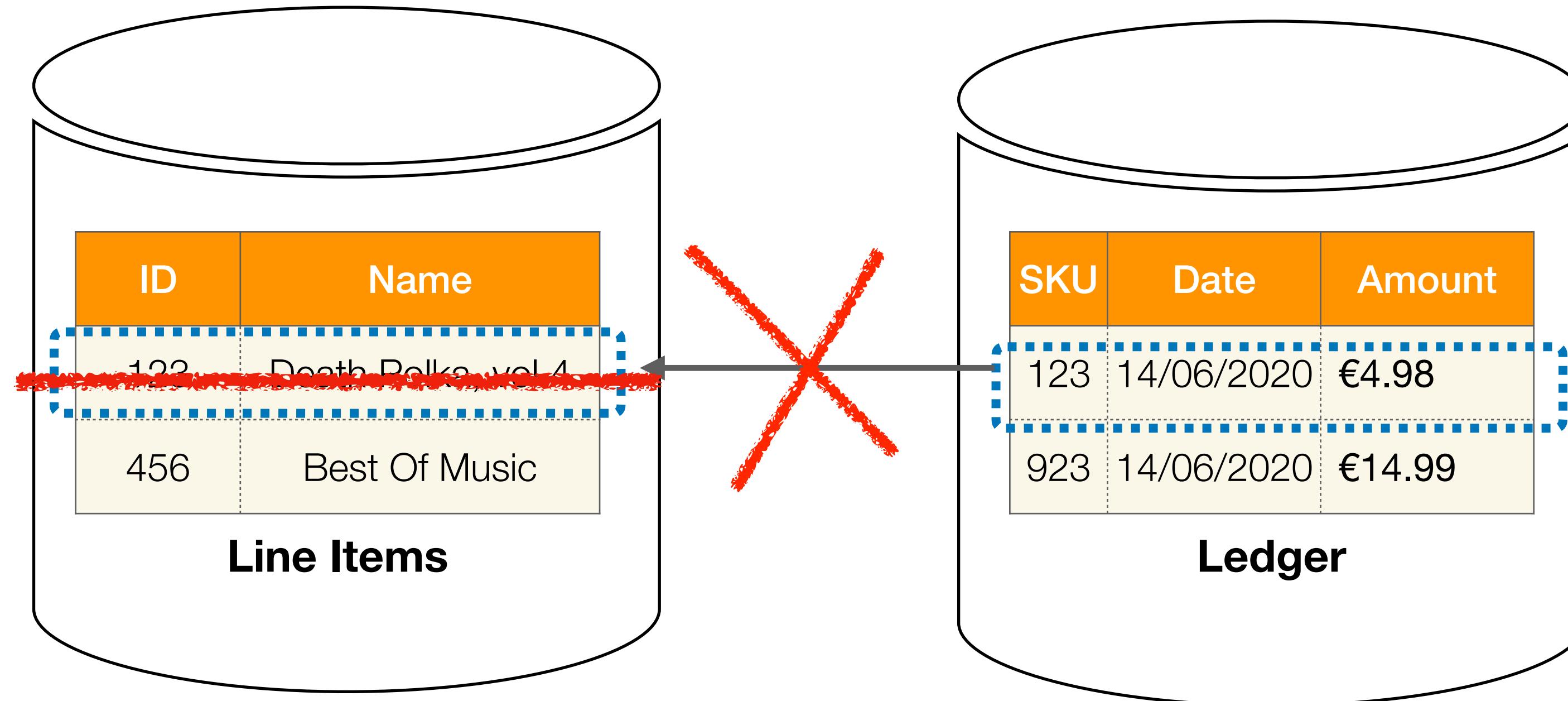
# REFERENTIAL INTEGRITY ACROSS DATABASES?



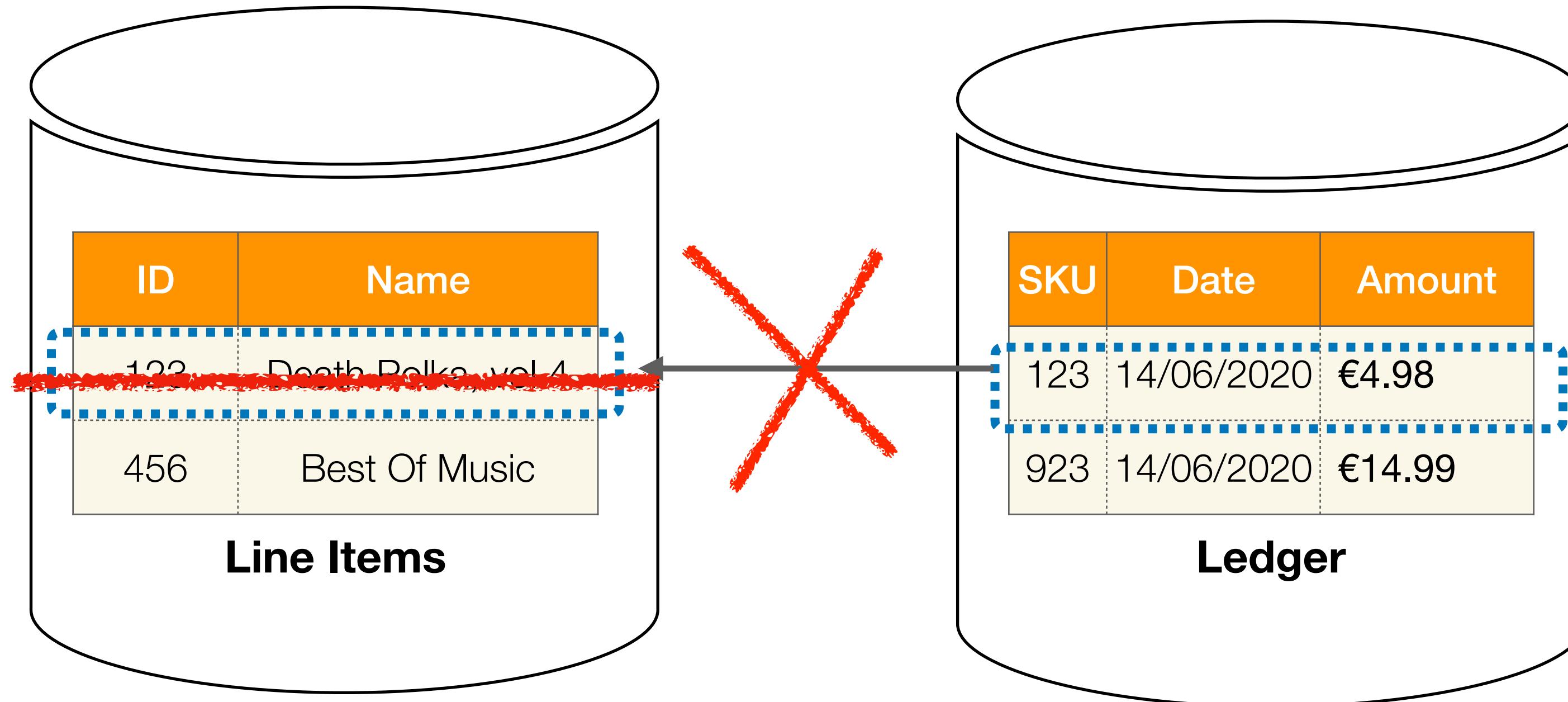
# REFERENTIAL INTEGRITY ACROSS DATABASES?



# REFERENTIAL INTEGRITY ACROSS DATABASES?

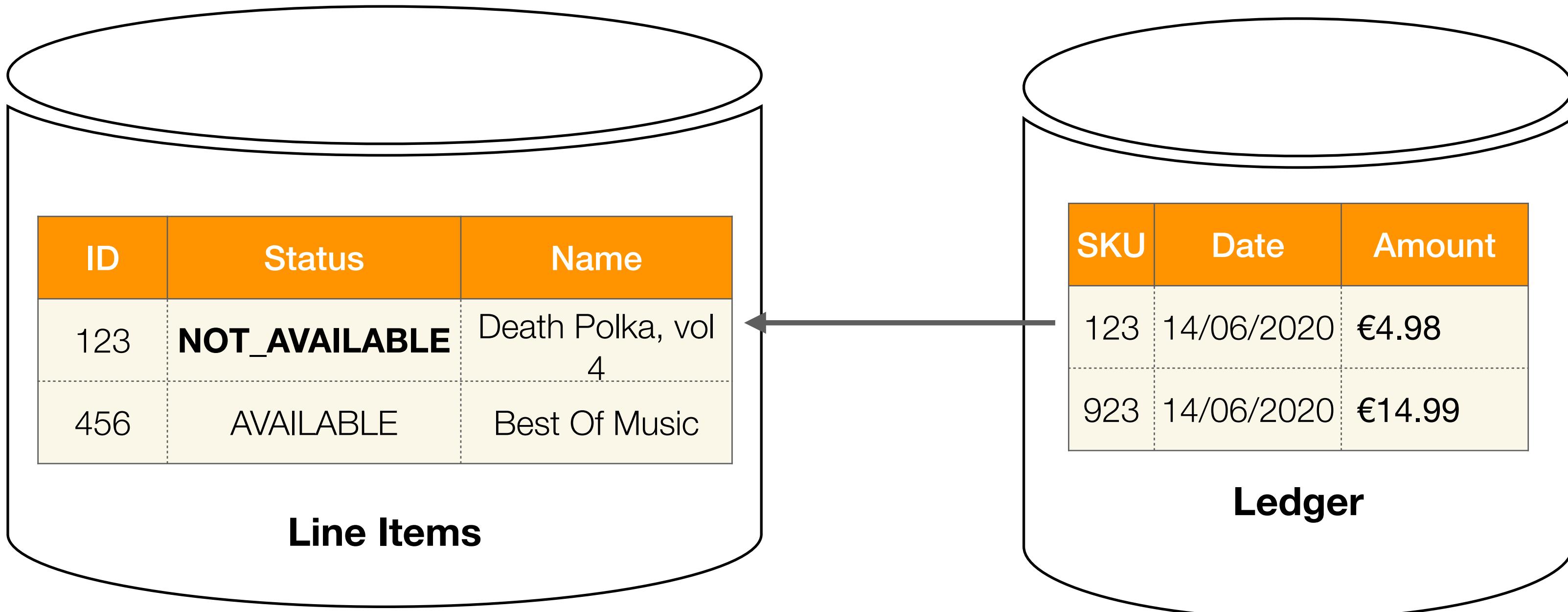


# REFERENTIAL INTEGRITY ACROSS DATABASES?

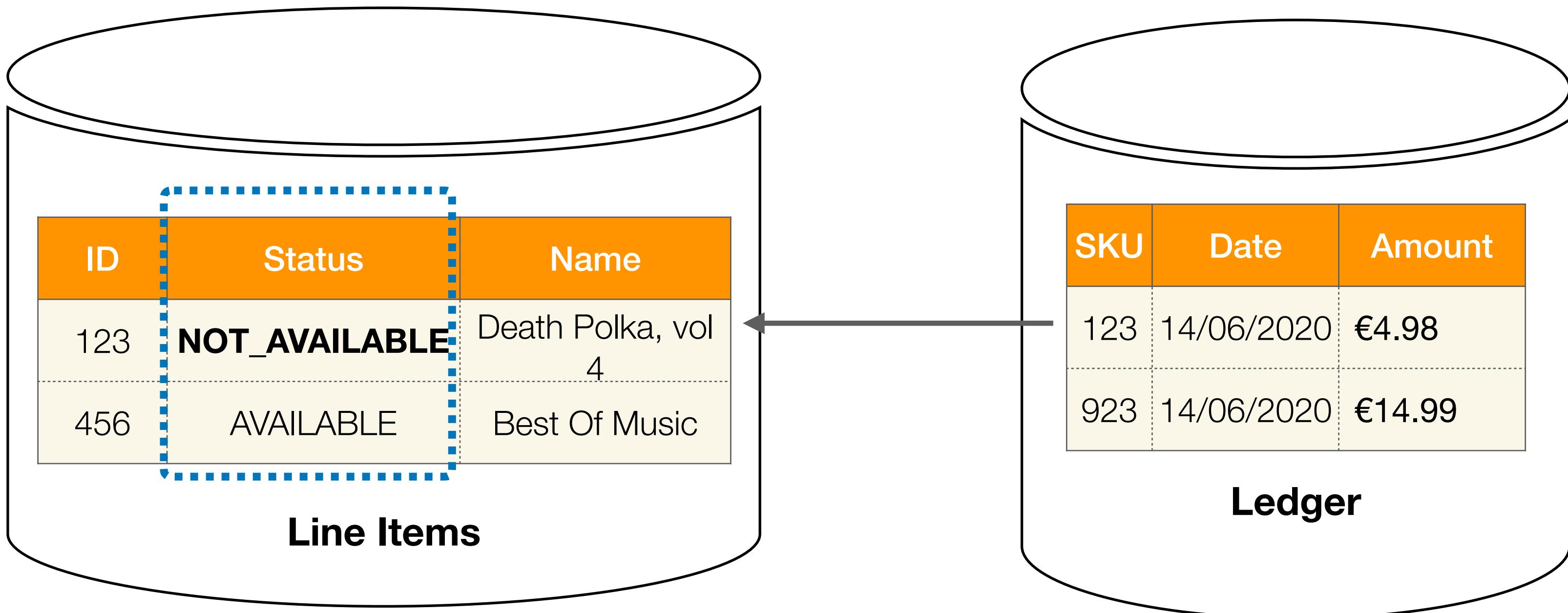


What are our options?

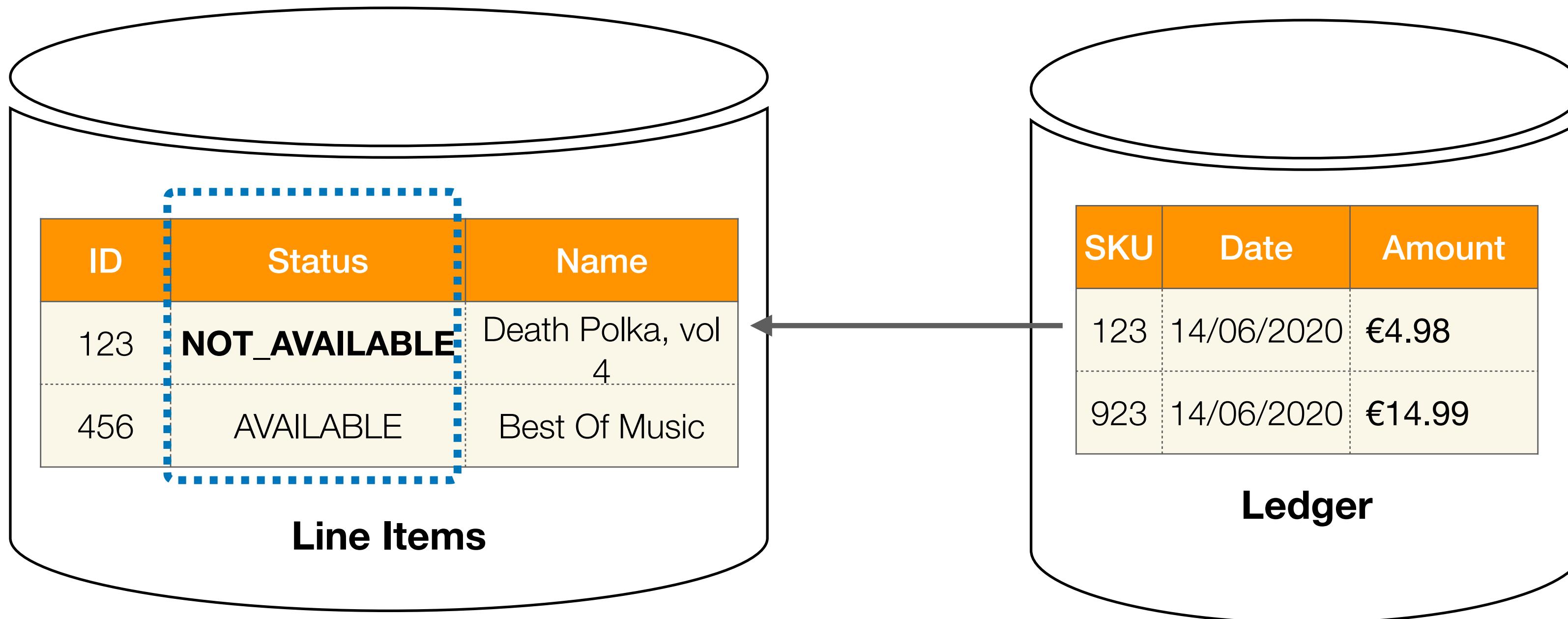
## SOFT DELETE



## SOFT DELETE



## SOFT DELETE



Data not removed, allowing internal references to continue to work

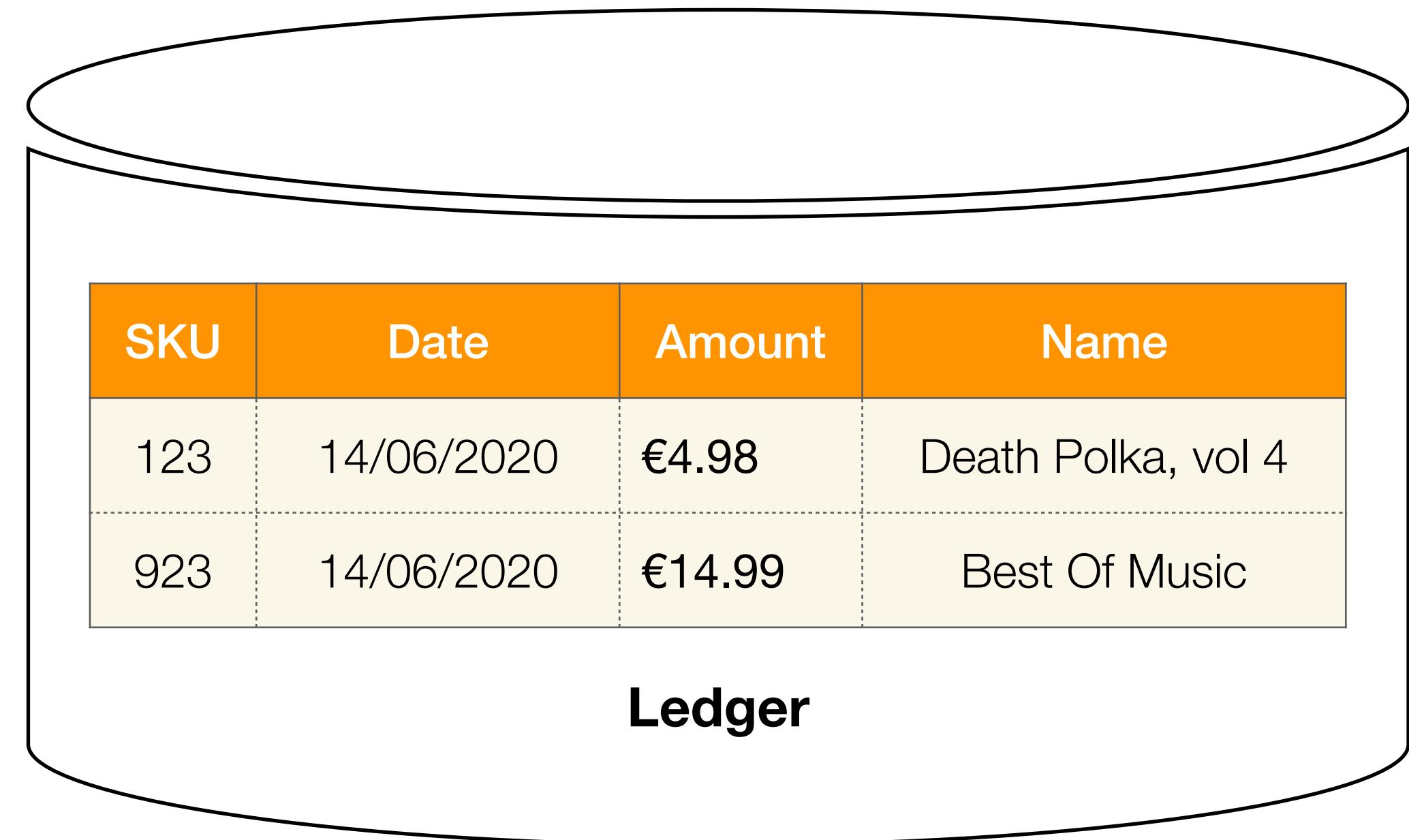
## COPY DATA



A cylinder representing the Line Items database. It contains a table with two columns: ID and Name.

ID	Name
123	Death Polka, vol 4
456	Best Of Music

**Line Items**



A cylinder representing the Ledger database. It contains a table with four columns: SKU, Date, Amount, and Name.

SKU	Date	Amount	Name
123	14/06/2020	€4.98	Death Polka, vol 4
923	14/06/2020	€14.99	Best Of Music

**Ledger**

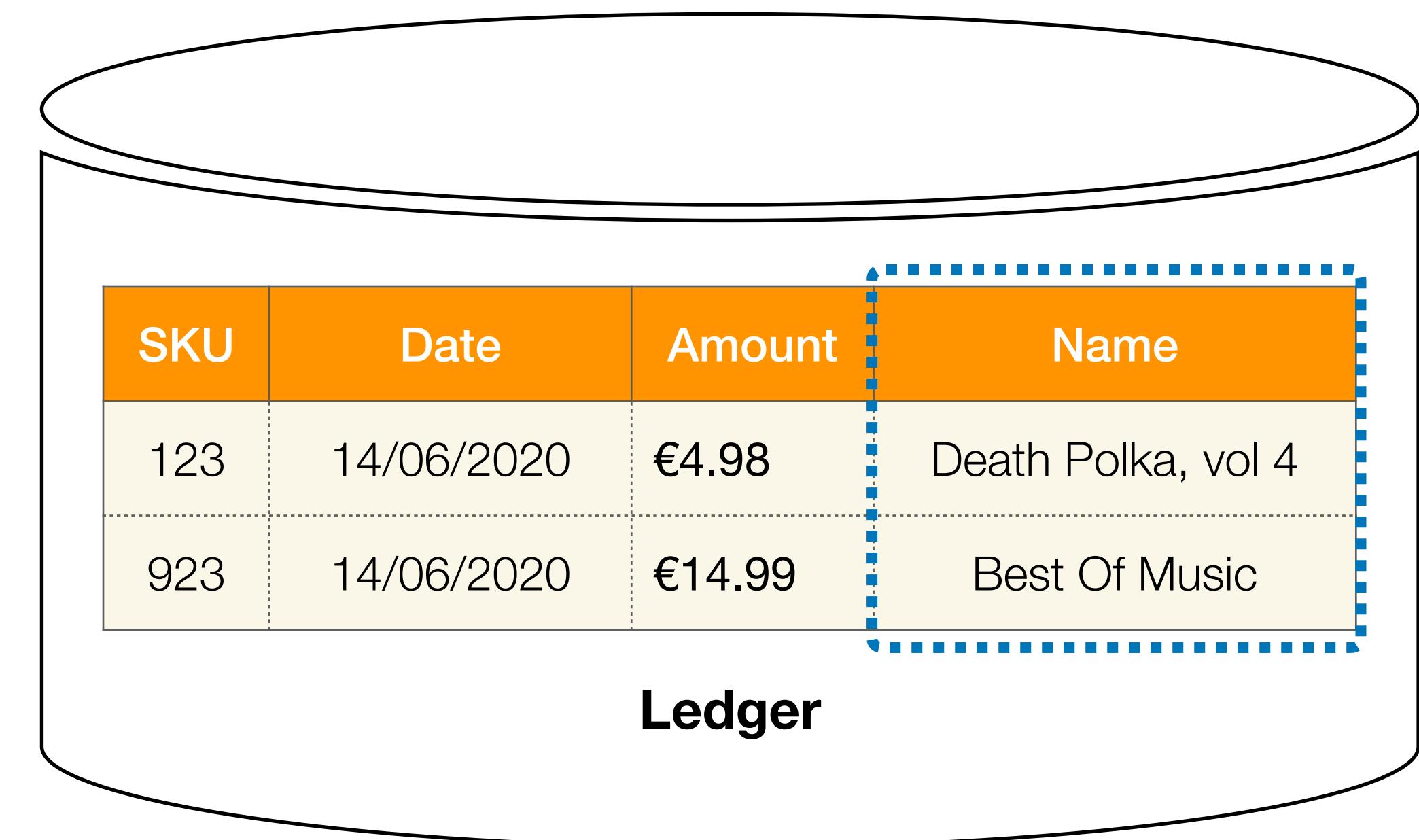
## COPY DATA



A cylinder representing the "Line Items" database. It contains a table with two columns: "ID" and "Name". The data rows are: ID 123, Name Death Polka, vol 4; and ID 456, Name Best Of Music.

ID	Name
123	Death Polka, vol 4
456	Best Of Music

**Line Items**

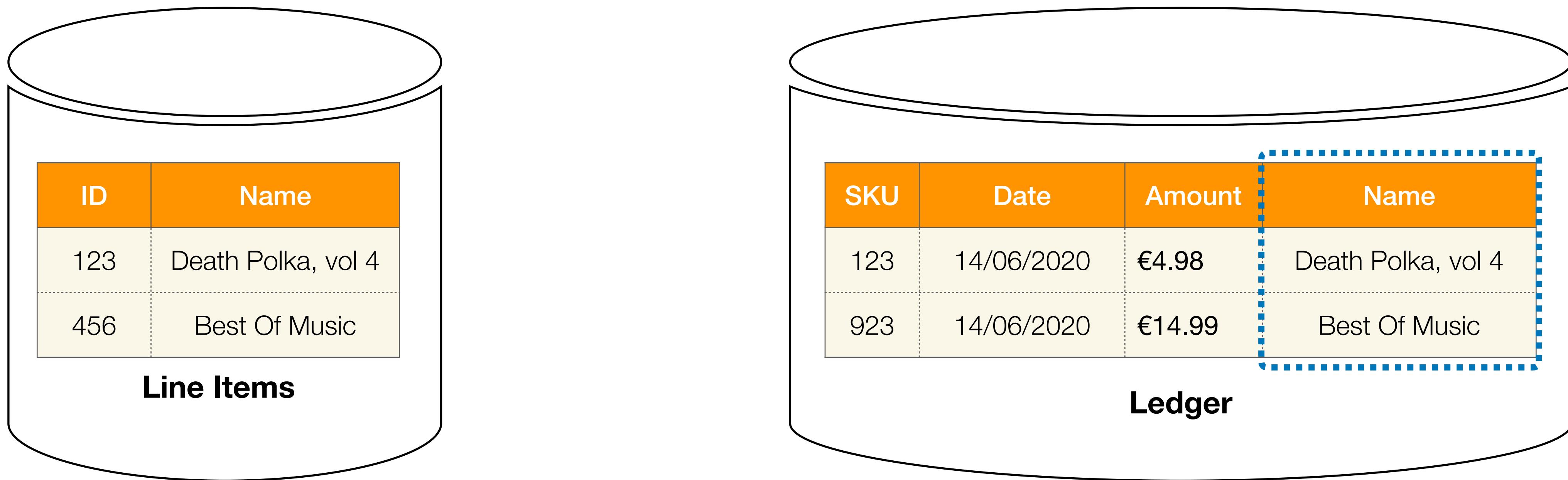


A cylinder representing the "Ledger" database. It contains a table with four columns: "SKU", "Date", "Amount", and "Name". The data rows are: SKU 123, Date 14/06/2020, Amount €4.98, Name Death Polka, vol 4; and SKU 923, Date 14/06/2020, Amount €14.99, Name Best Of Music. A blue dashed box highlights the "Name" column.

SKU	Date	Amount	Name
123	14/06/2020	€4.98	Death Polka, vol 4
923	14/06/2020	€14.99	Best Of Music

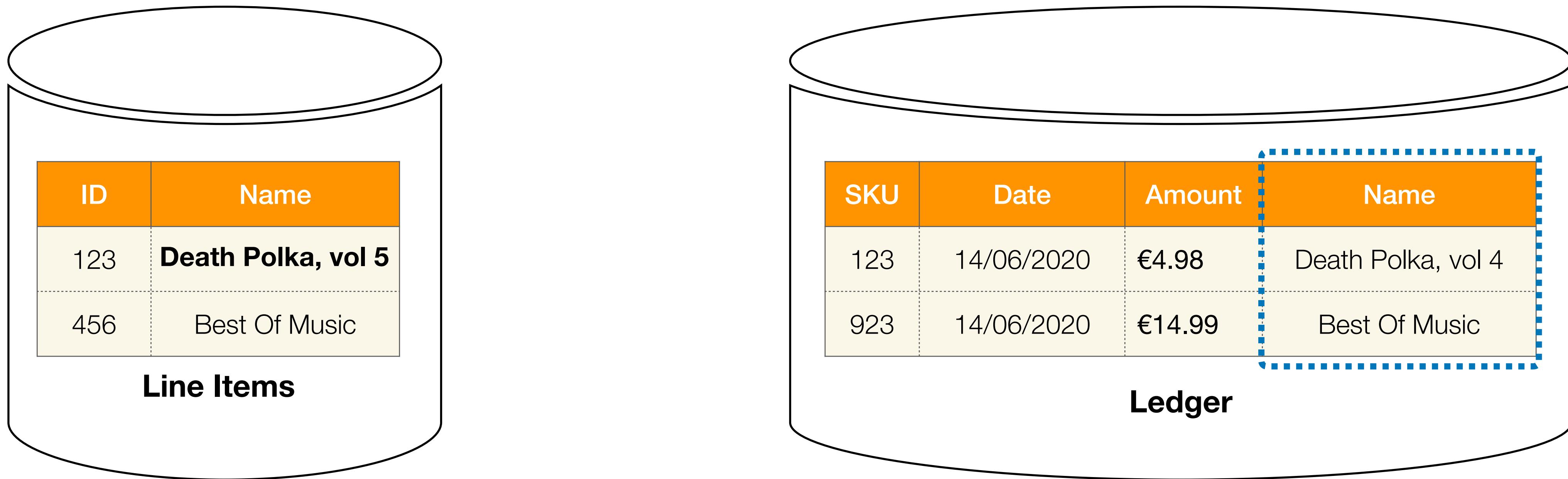
**Ledger**

## COPY DATA



Avoids the need for the join if the name is the only thing needed

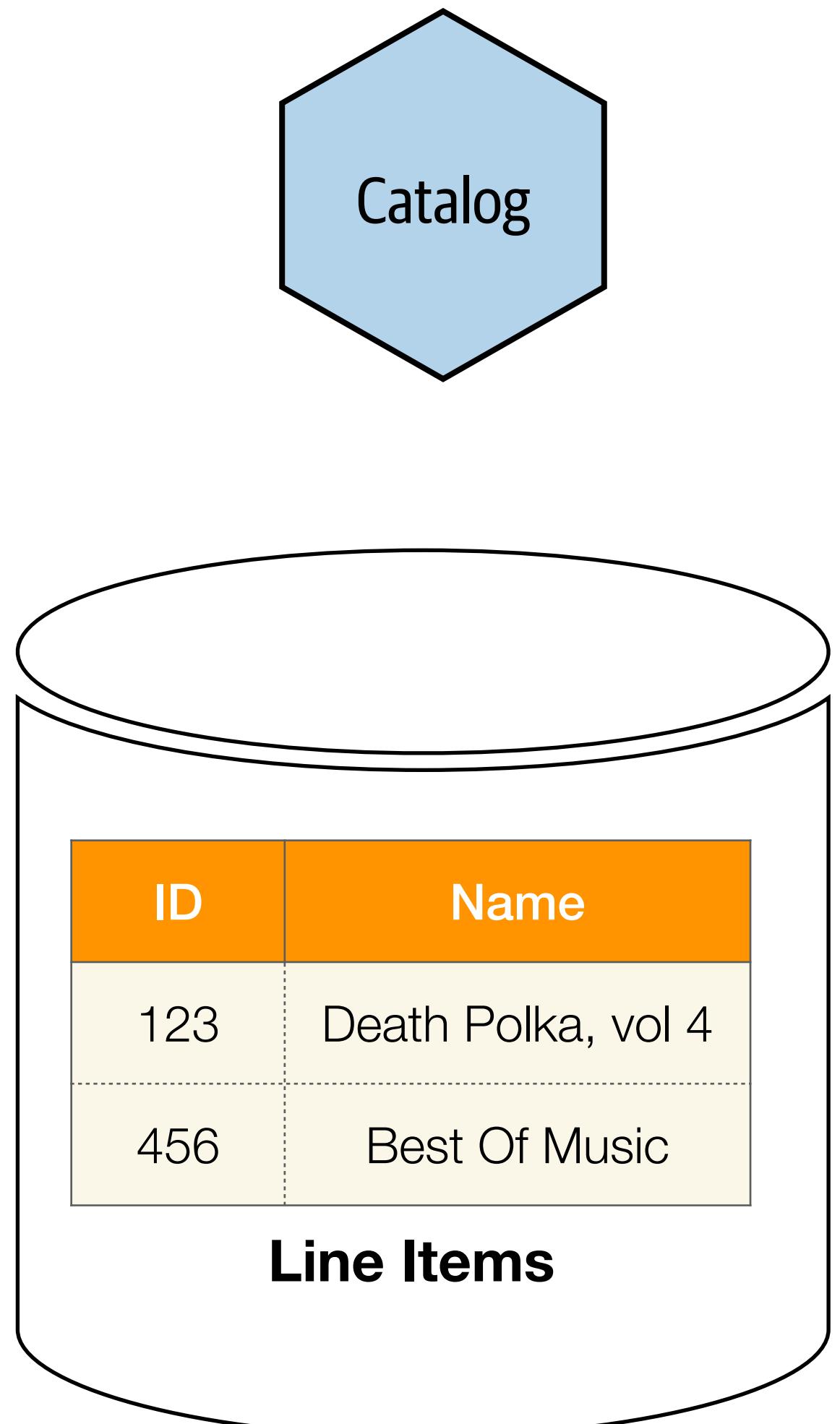
## COPY DATA



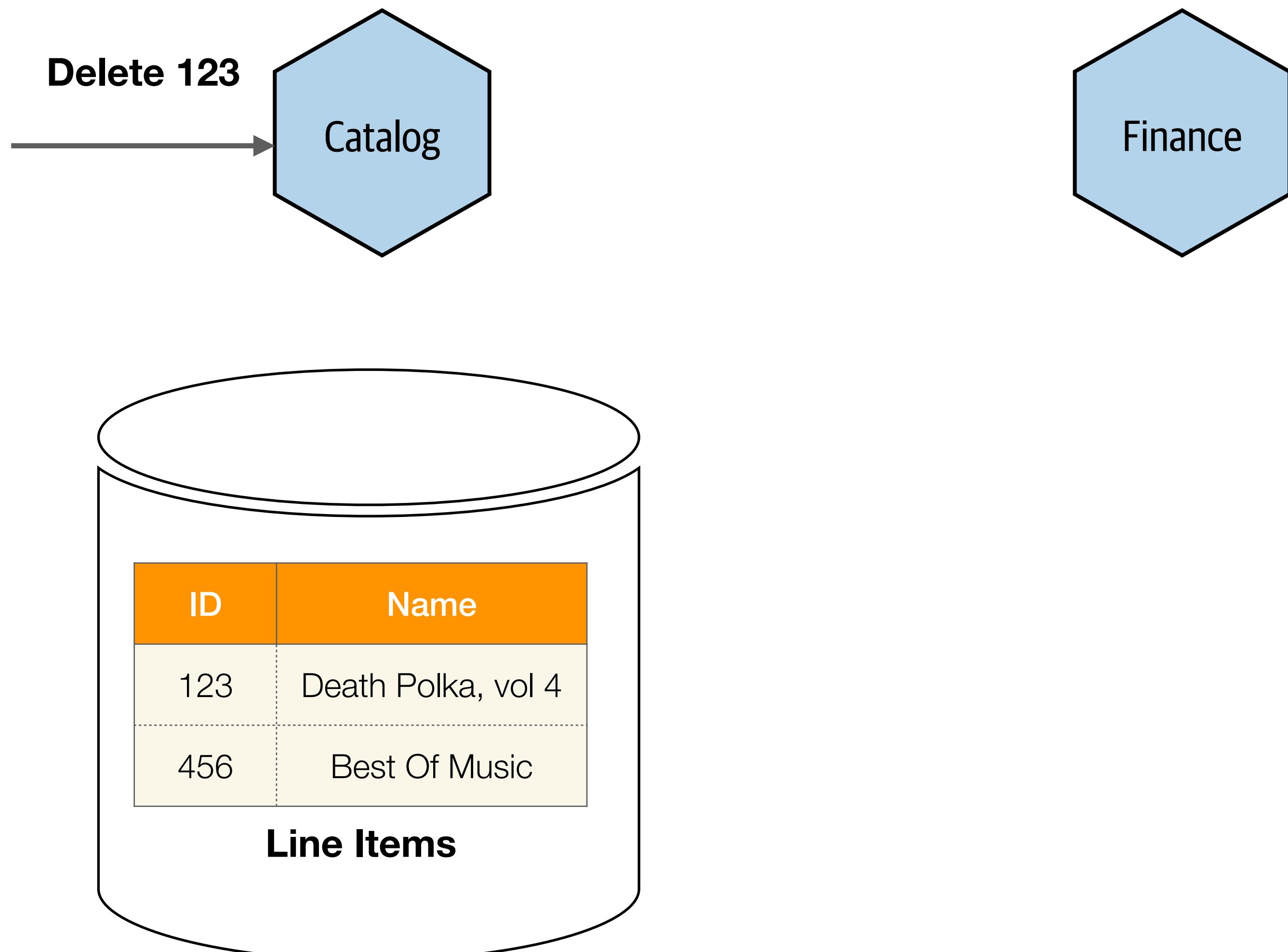
Avoids the need for the join if the name is the only thing needed

But what about if the source data changes?

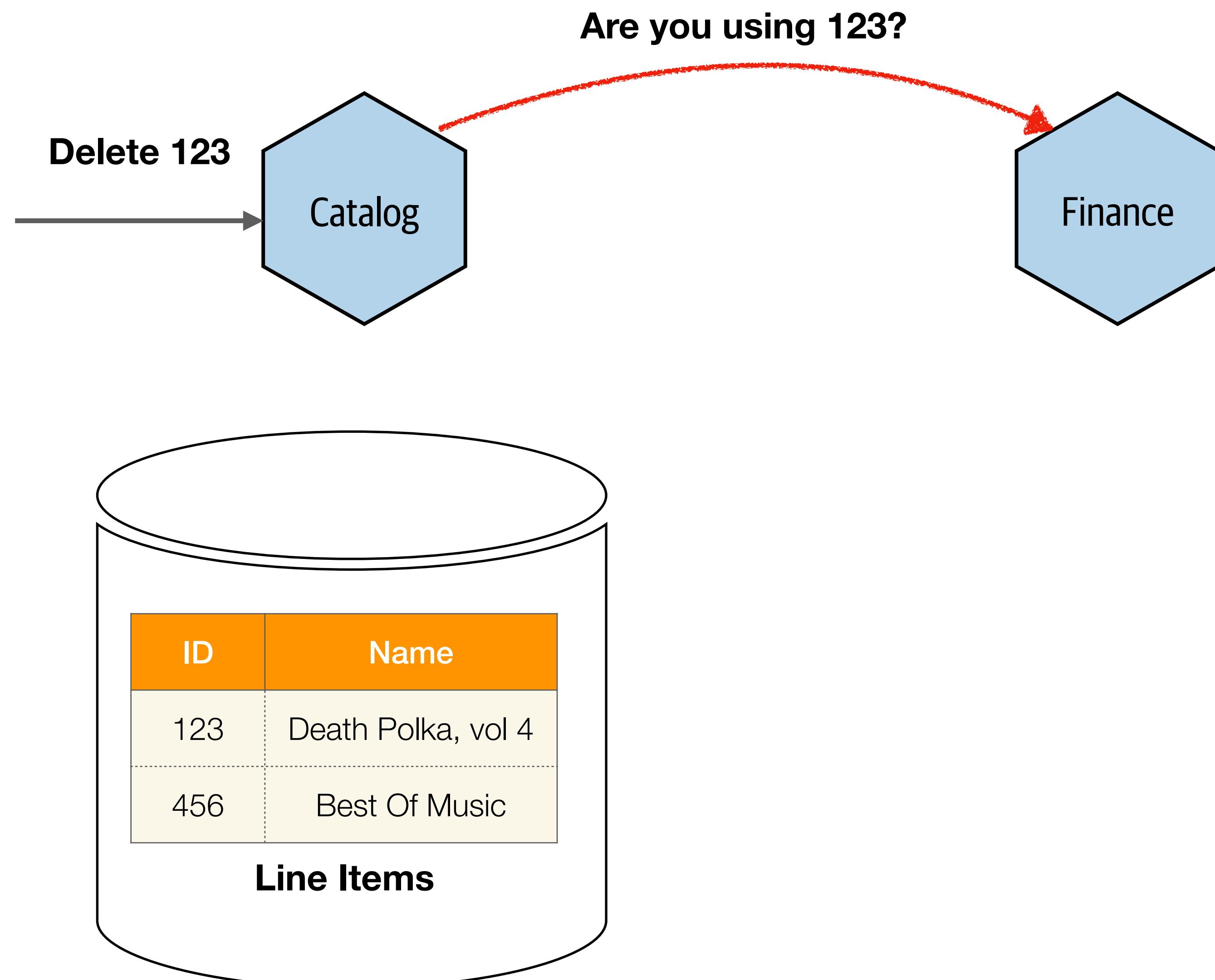
## CHECK FOR USE BEFORE DELETE?



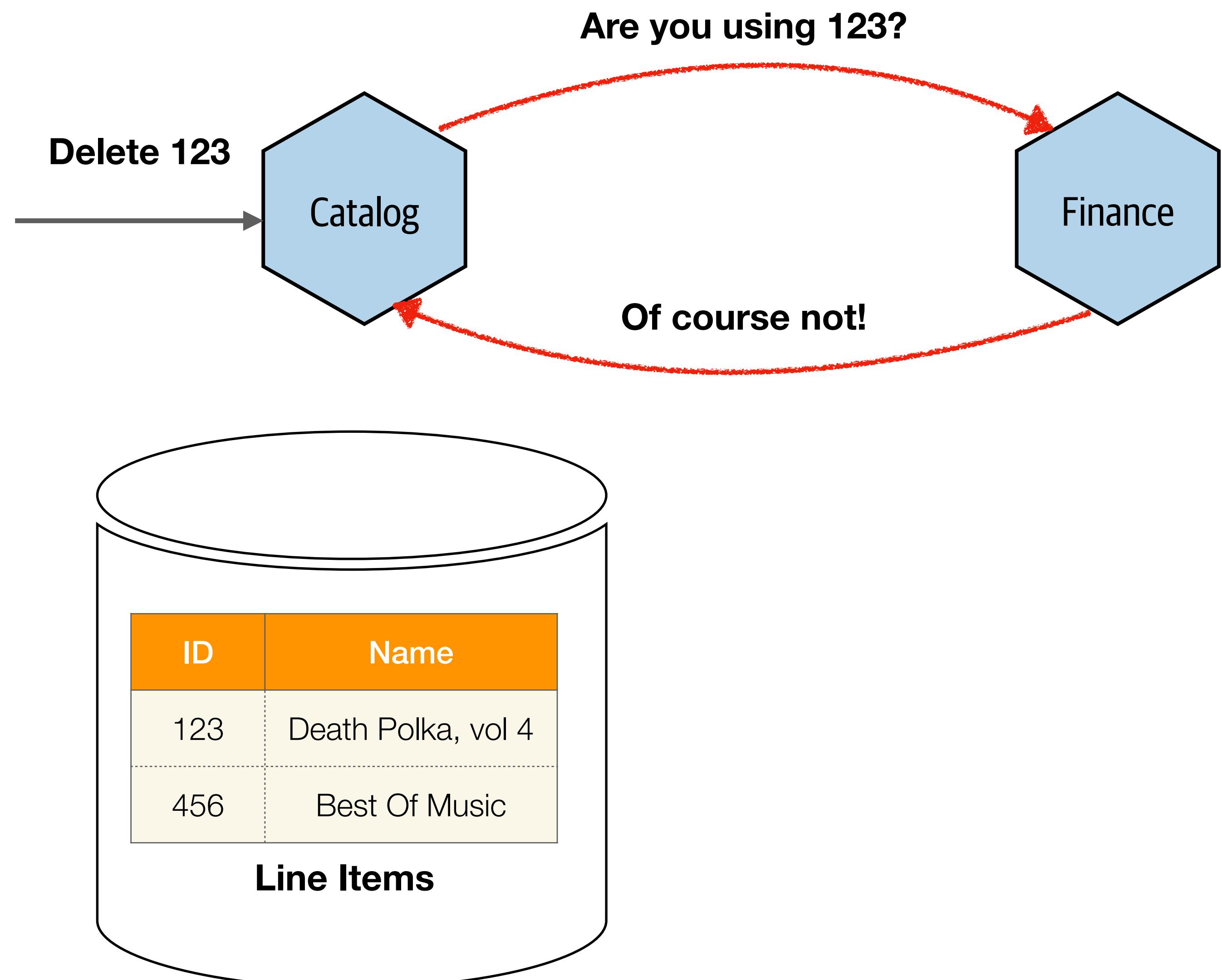
## CHECK FOR USE BEFORE DELETE?



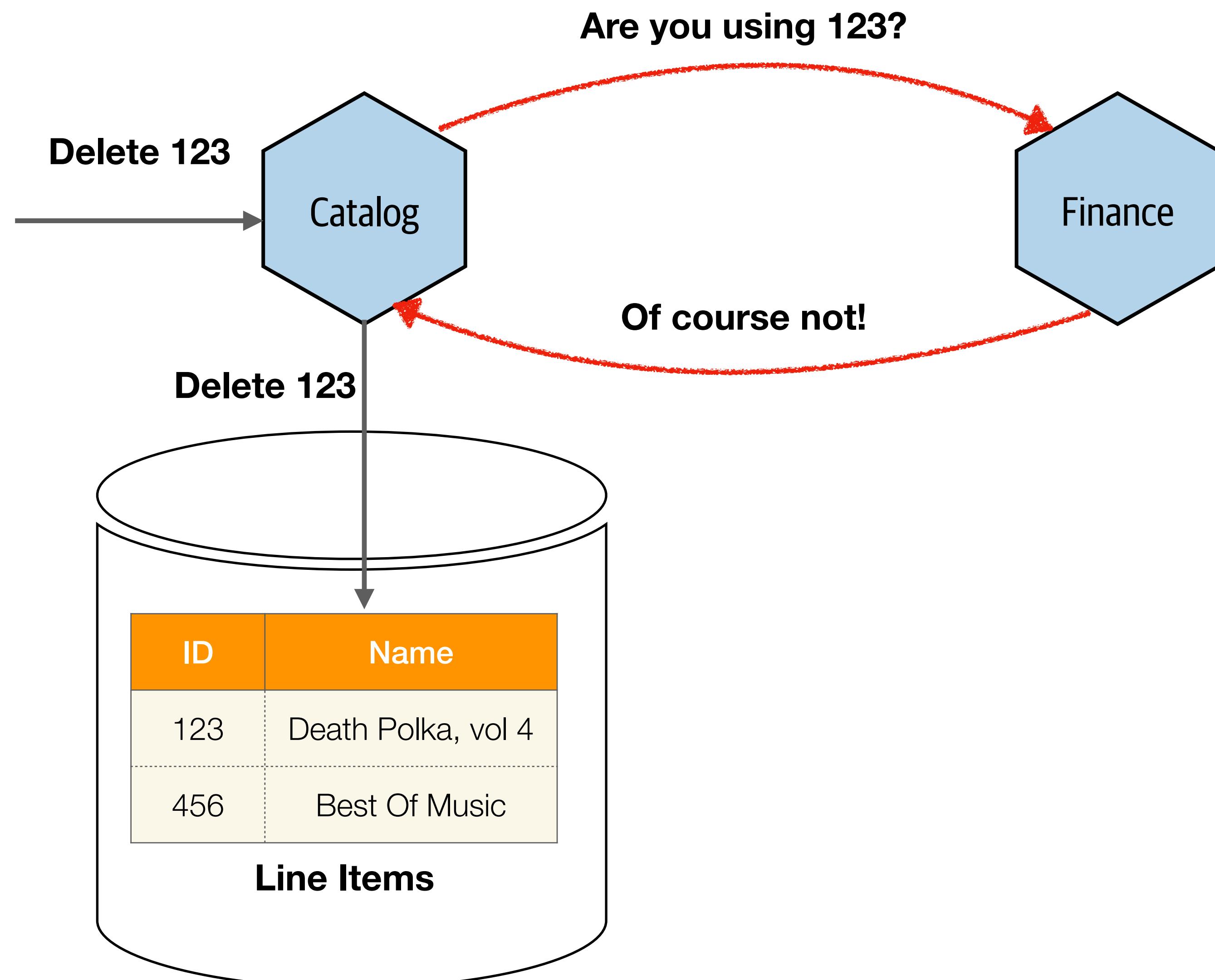
## CHECK FOR USE BEFORE DELETE?



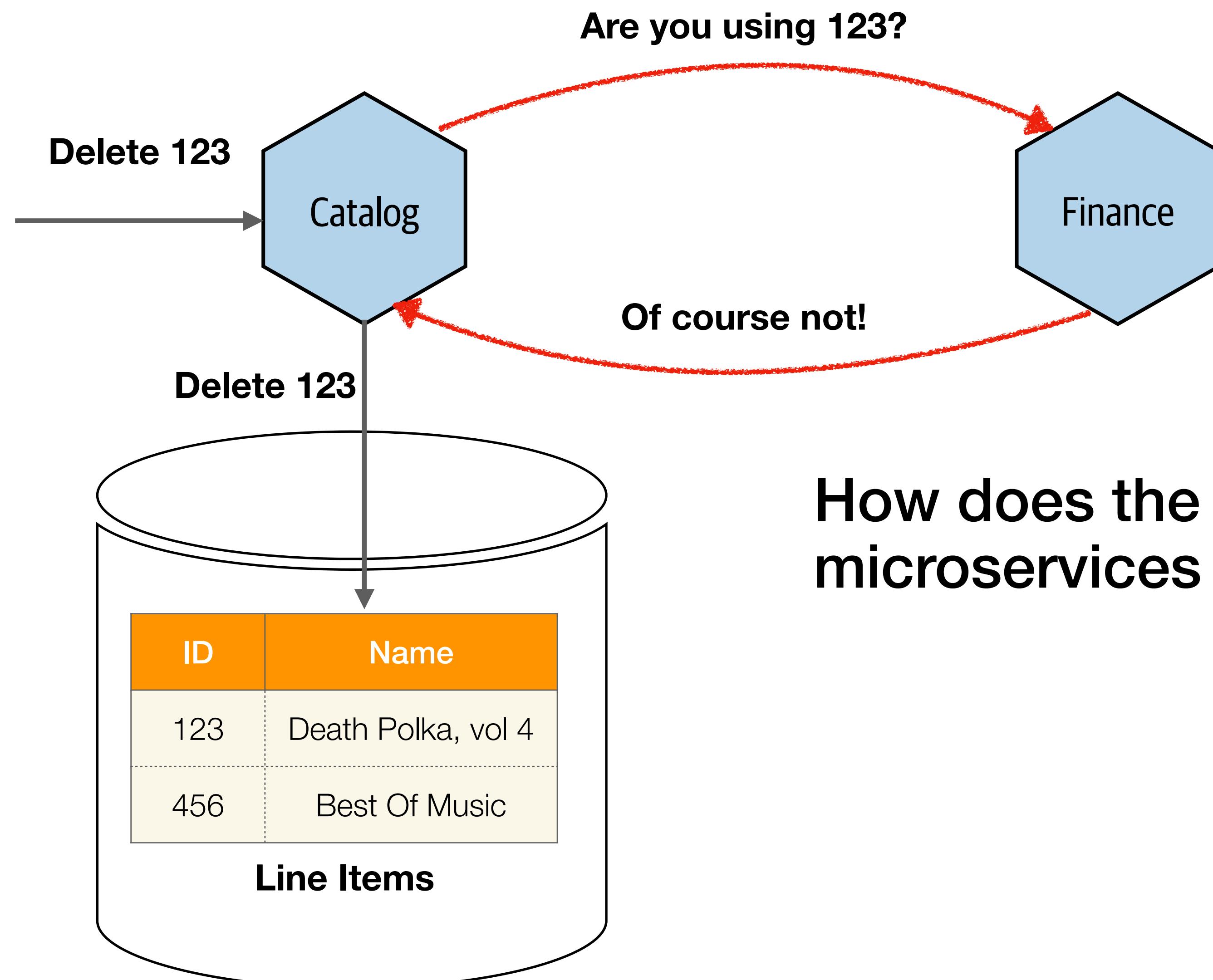
## CHECK FOR USE BEFORE DELETE?



## CHECK FOR USE BEFORE DELETE?

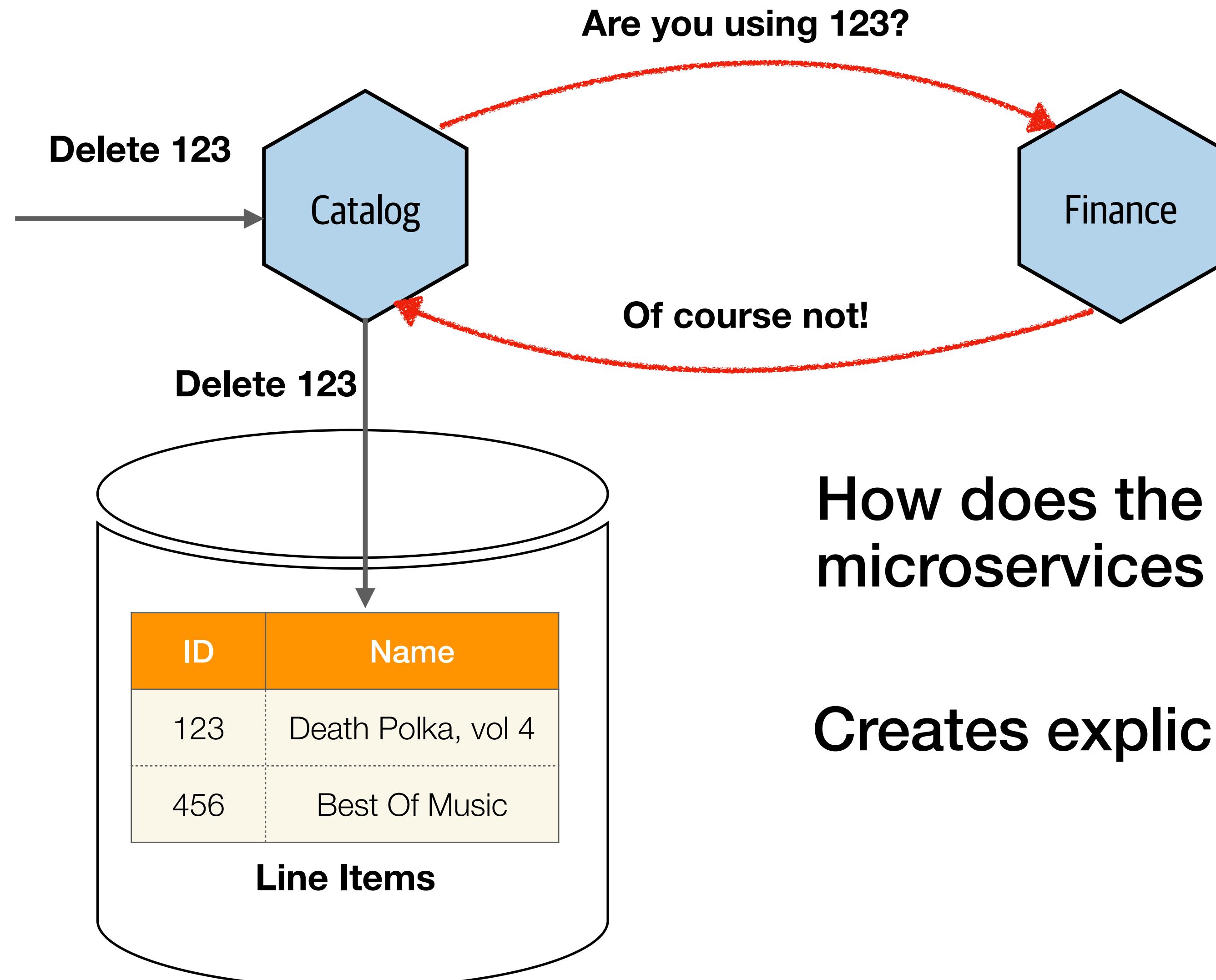


## CHECK FOR USE BEFORE DELETE?



How does the catalog know which microservices to ask?

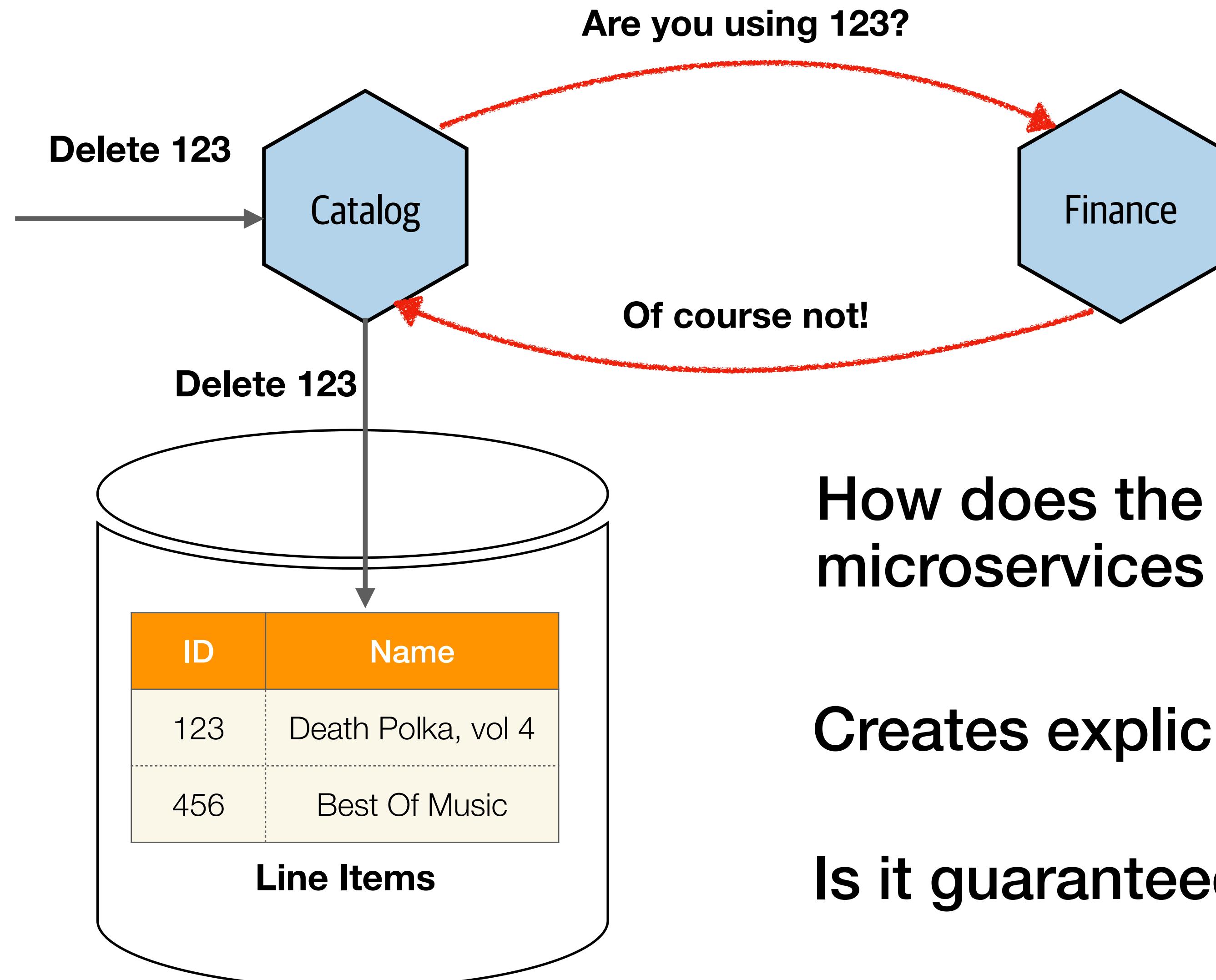
## CHECK FOR USE BEFORE DELETE?



How does the catalog know which microservices to ask?

Creates explicit circular dependencies

## CHECK FOR USE BEFORE DELETE?



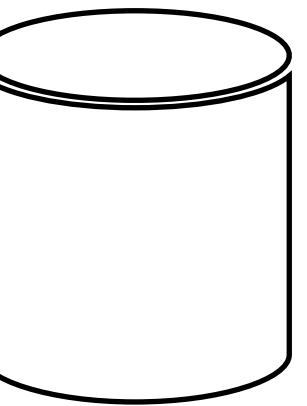
How does the catalog know which microservices to ask?

Creates explicit circular dependencies

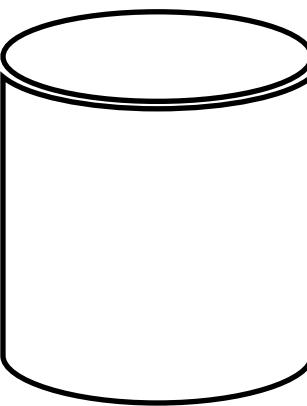
Is it guaranteed to work?

# LOCKING REQUIRED?

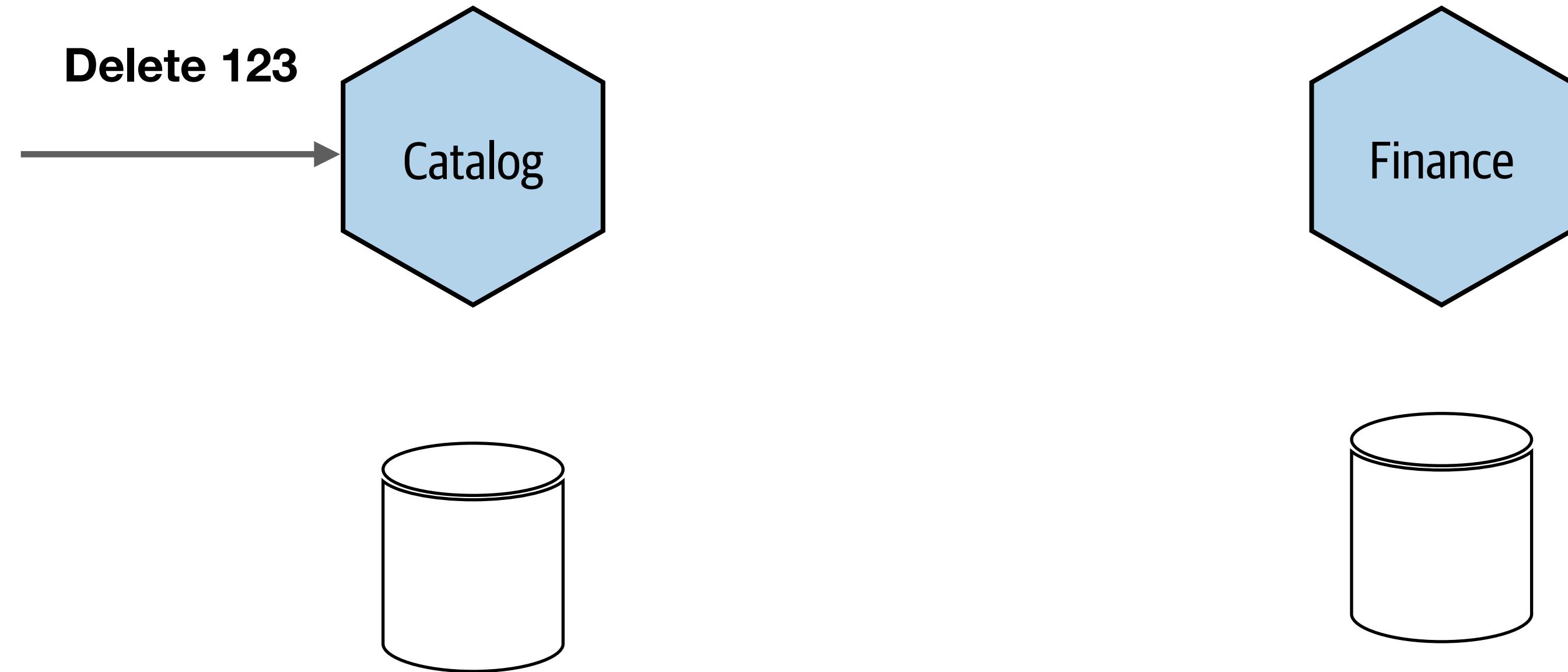
Catalog



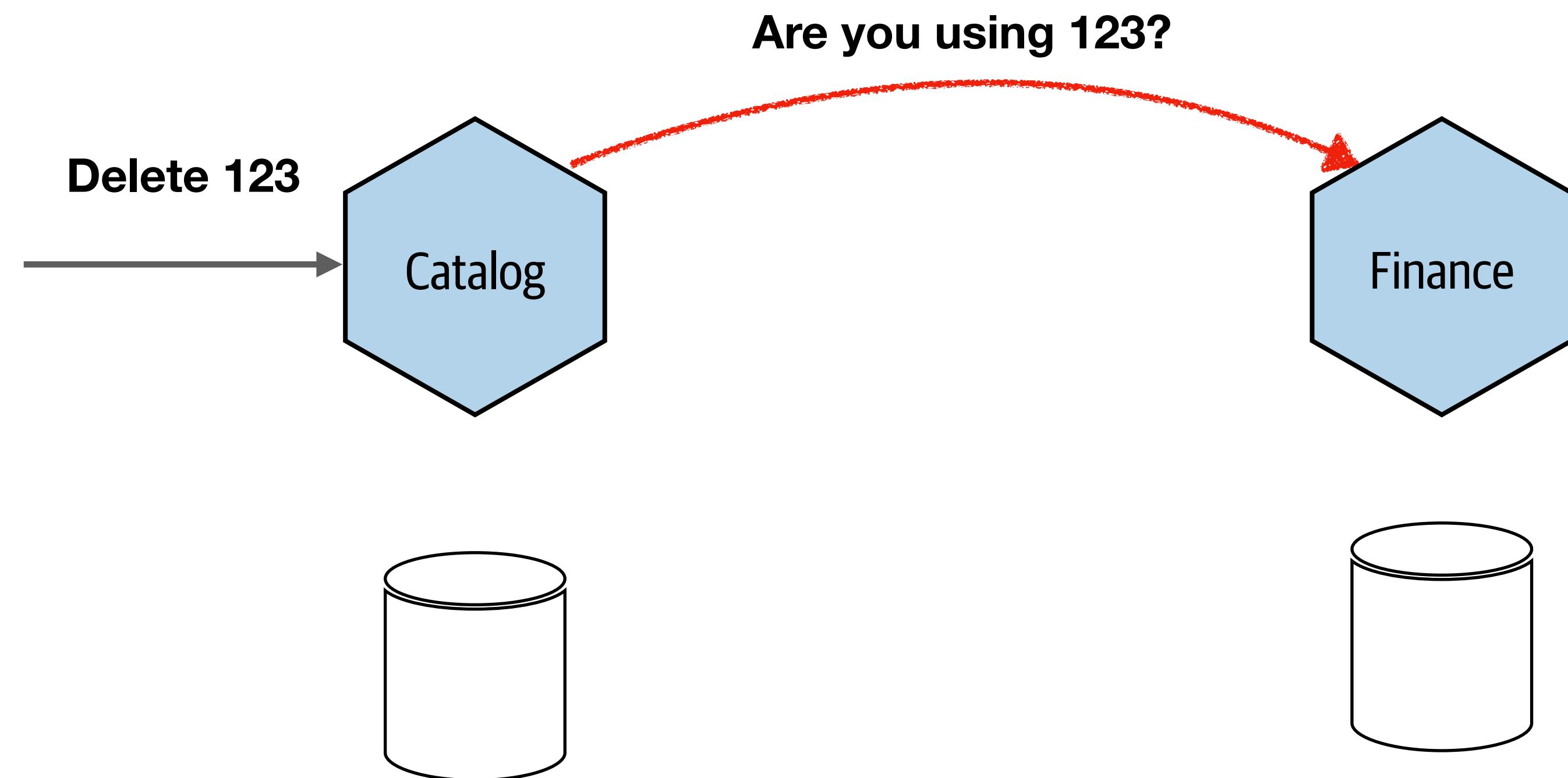
Finance



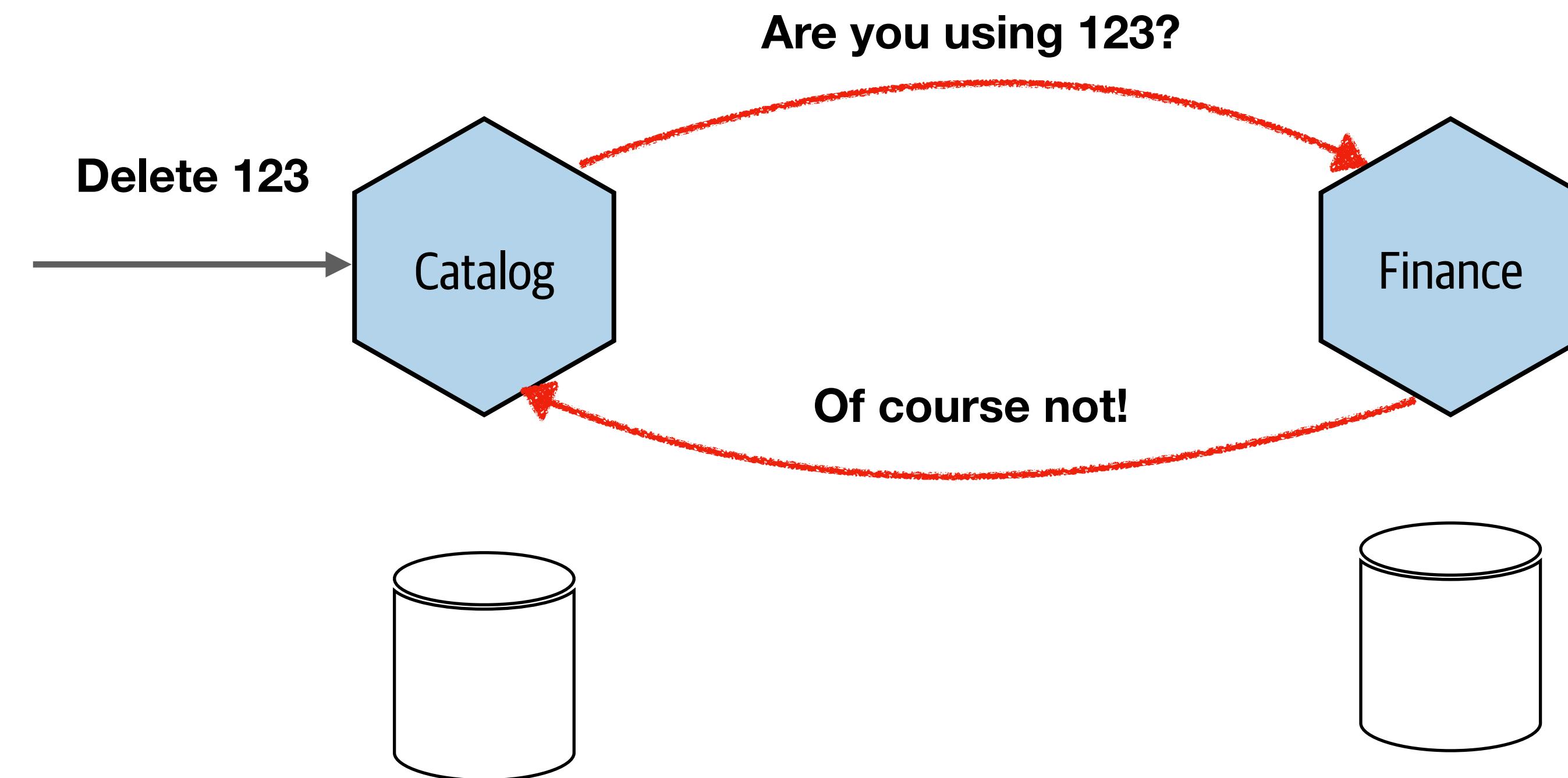
## LOCKING REQUIRED?



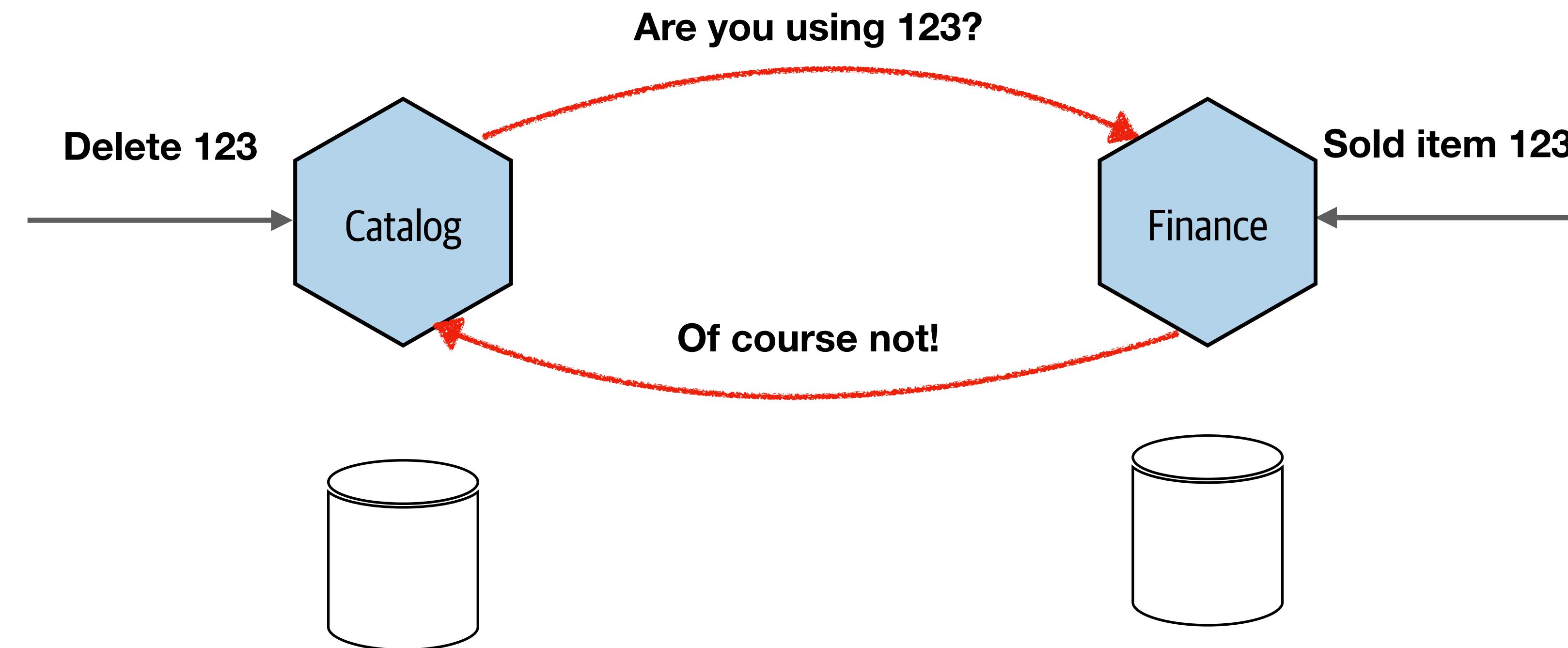
## LOCKING REQUIRED?



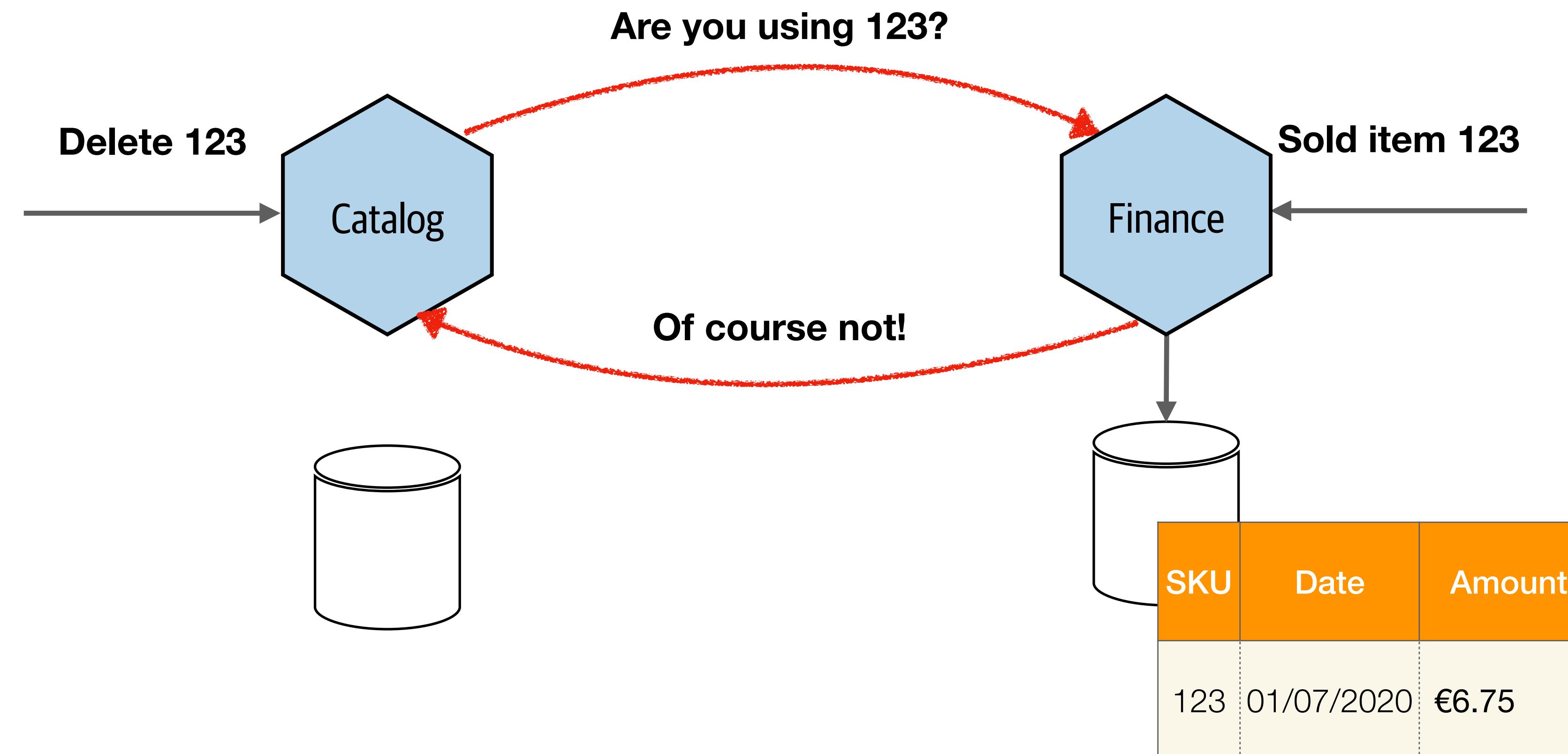
## LOCKING REQUIRED?



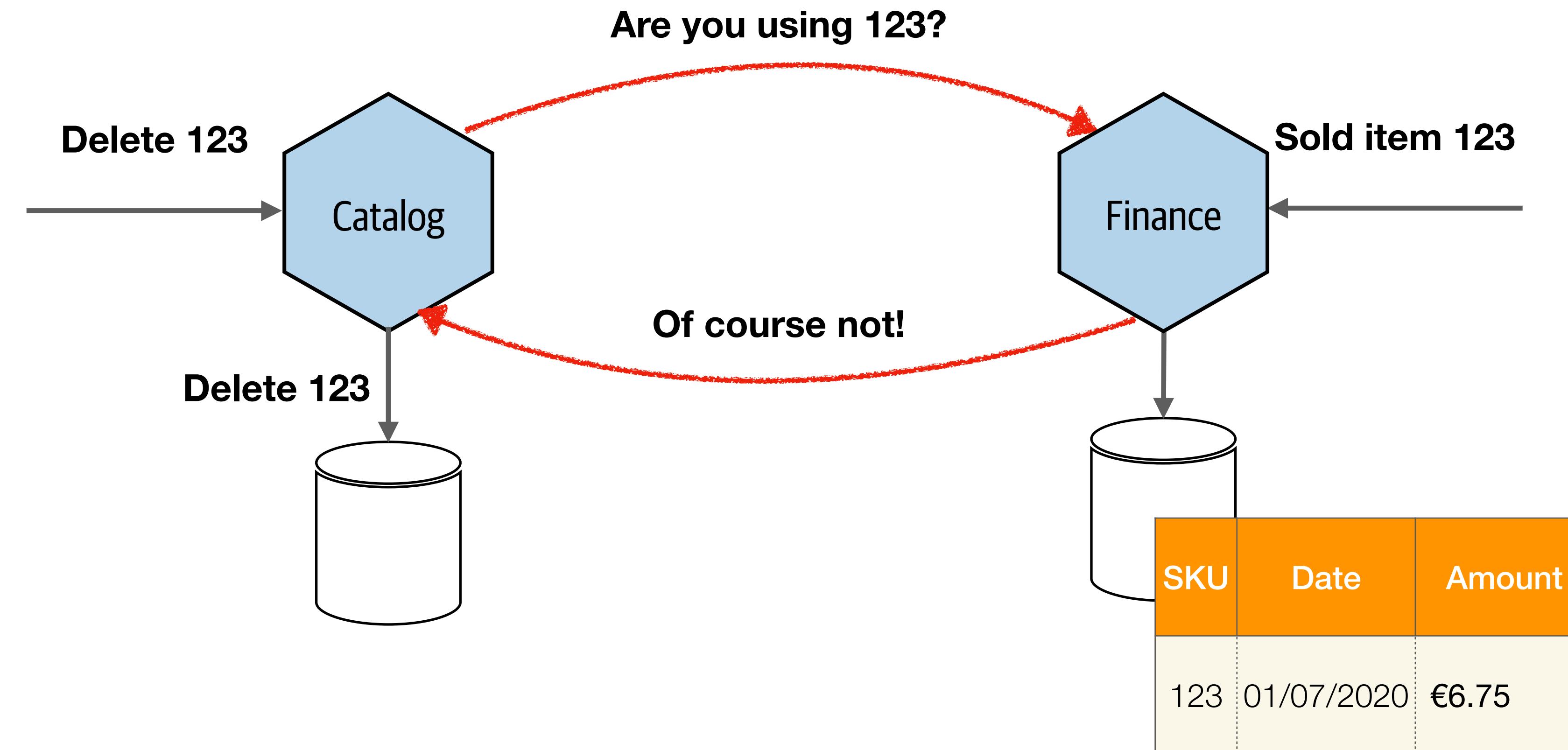
## LOCKING REQUIRED?



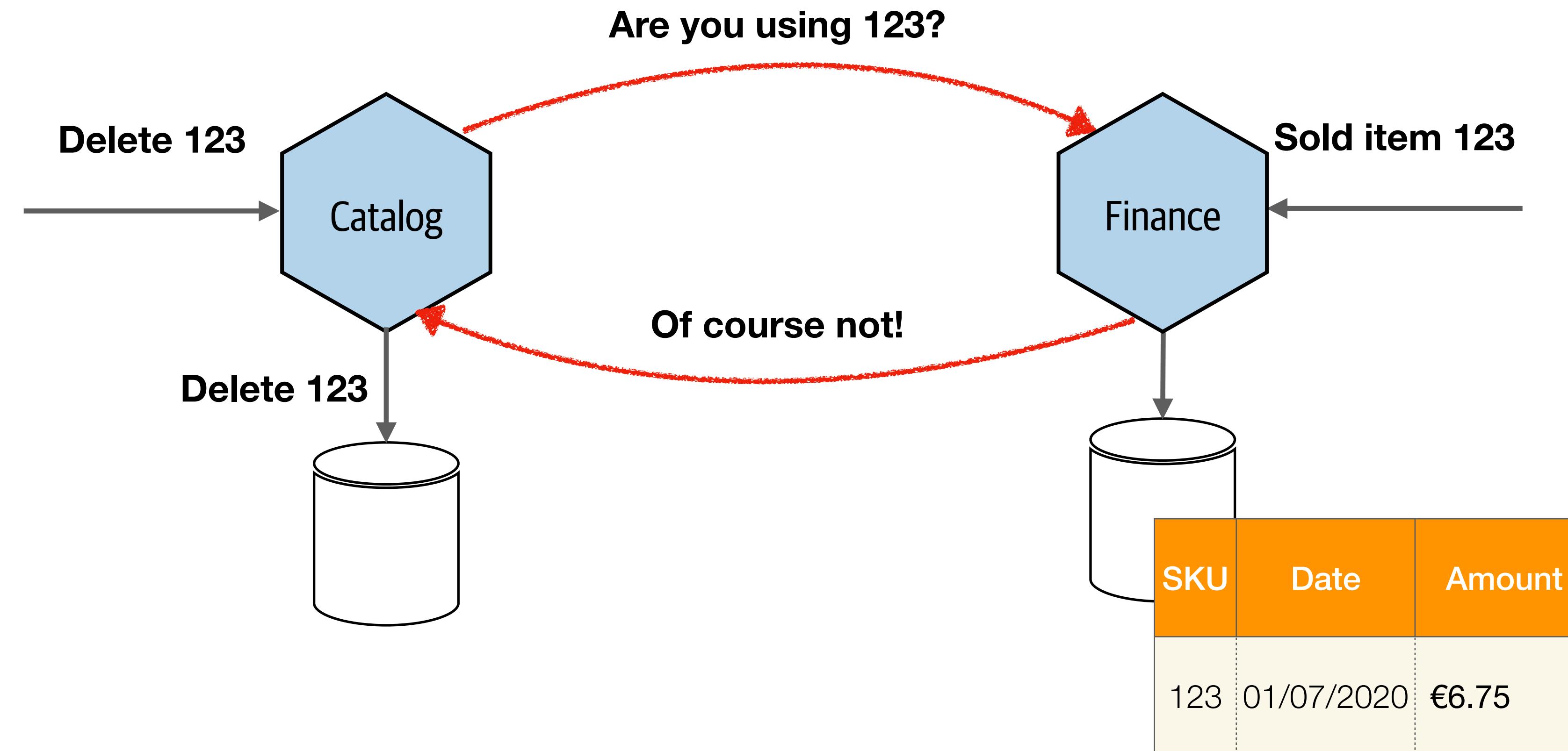
## LOCKING REQUIRED?



## LOCKING REQUIRED?

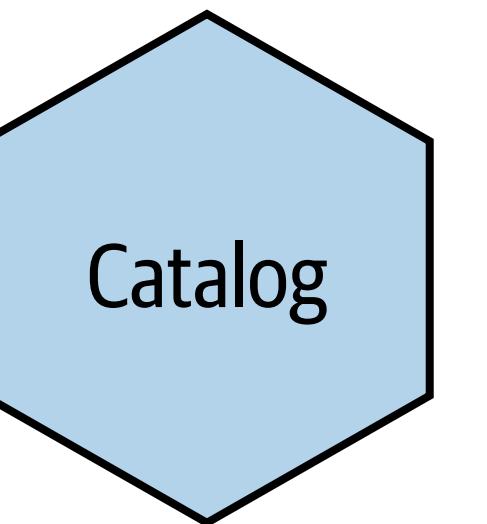


## LOCKING REQUIRED?

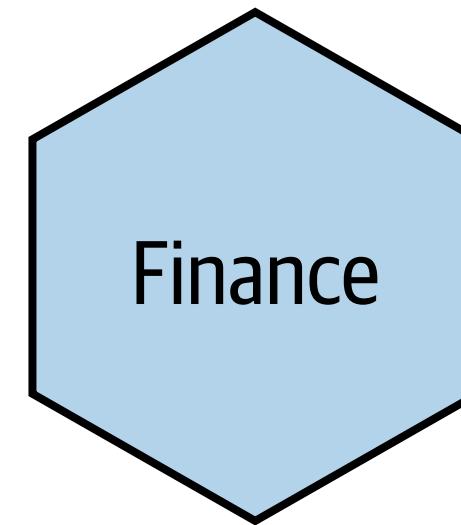


You need to lock the ledger table if you want to be certain that no references to 123 exist

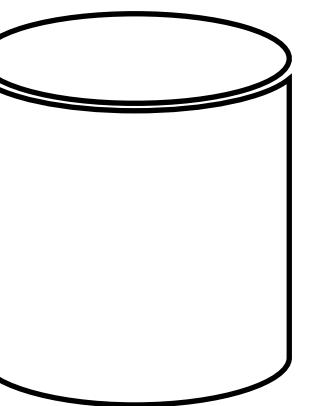
# CASCADING DELETES?



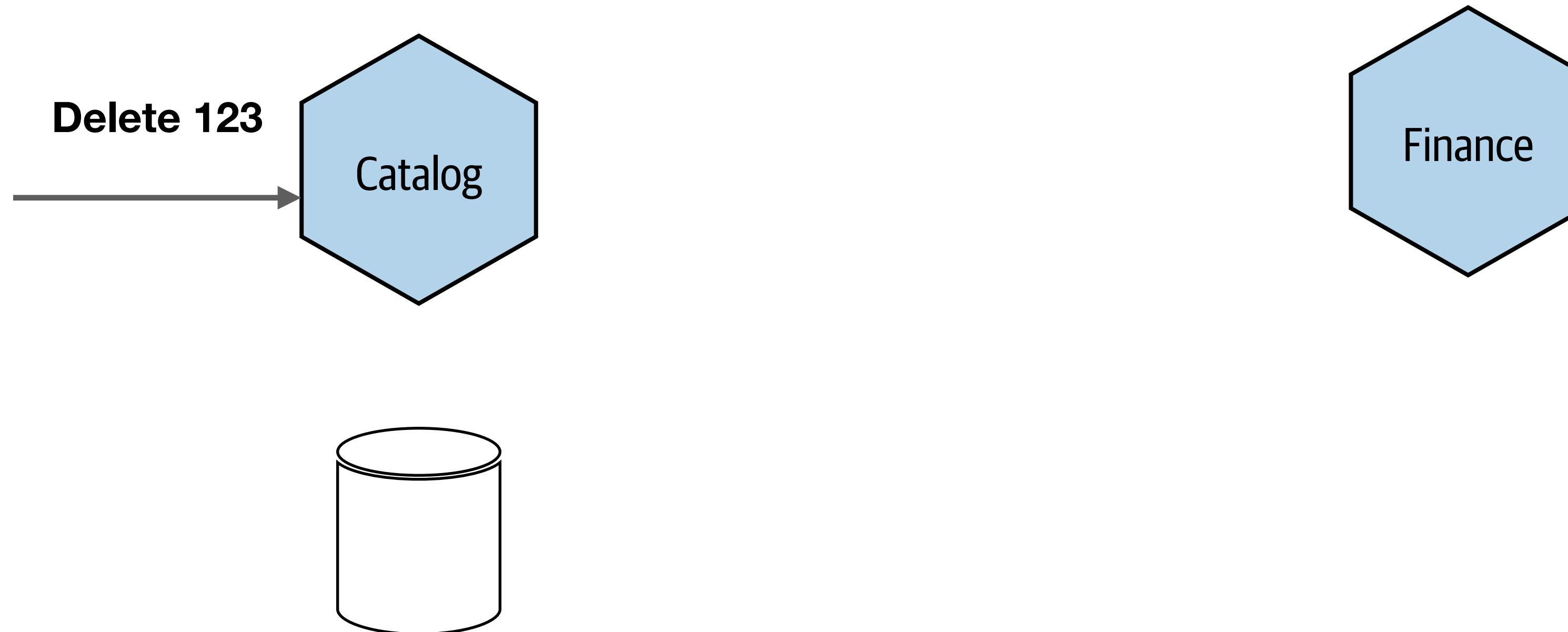
Catalog



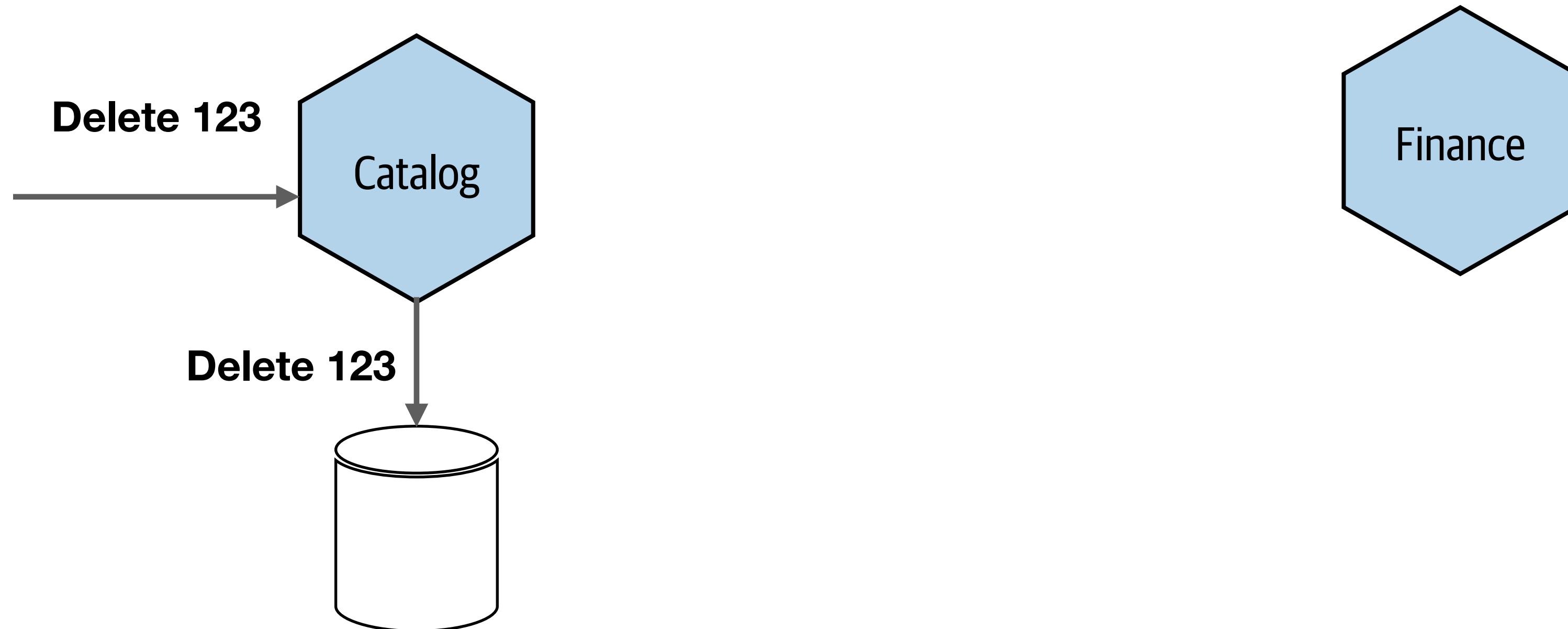
Finance



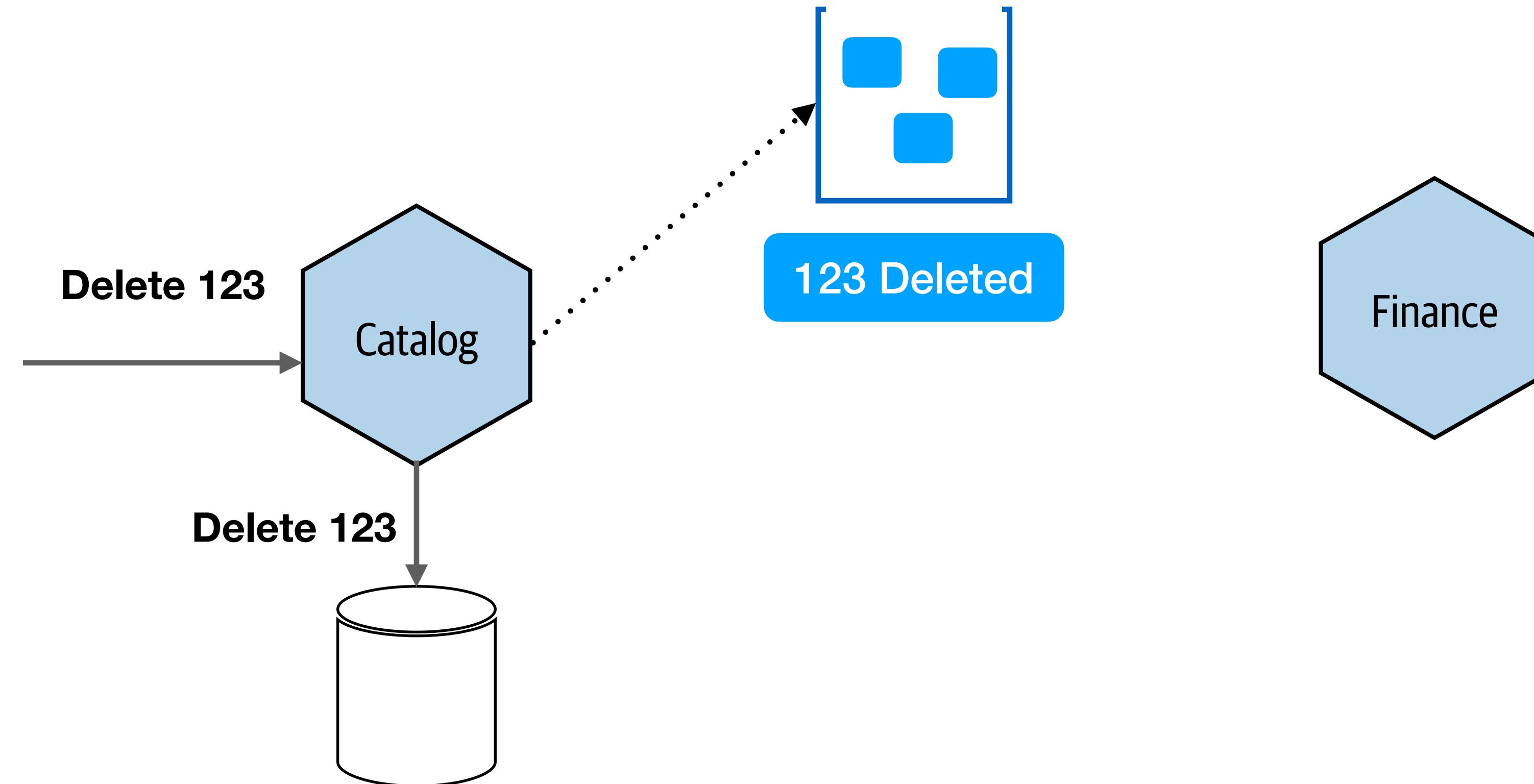
## CASCADING DELETES?



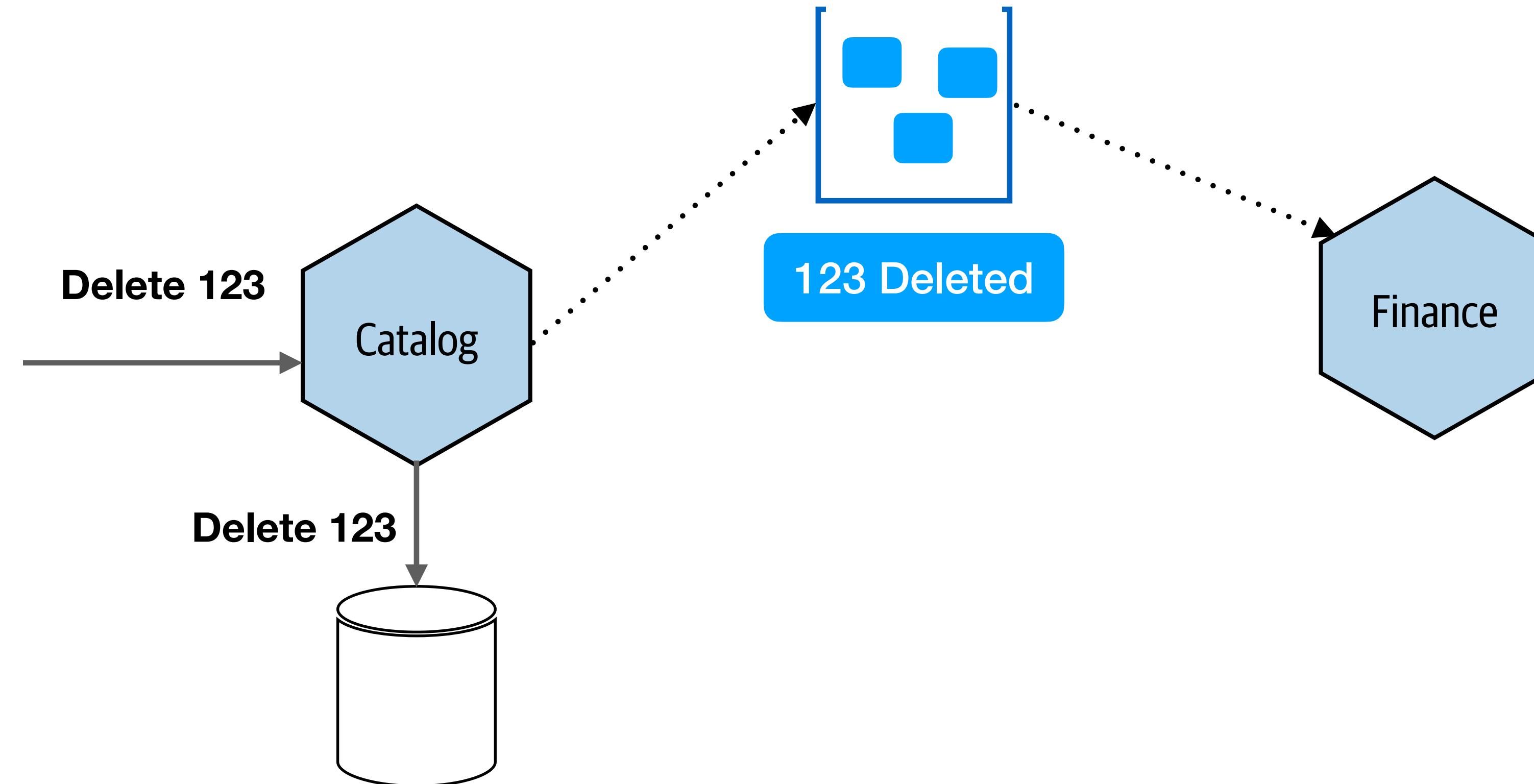
## CASCADING DELETES?



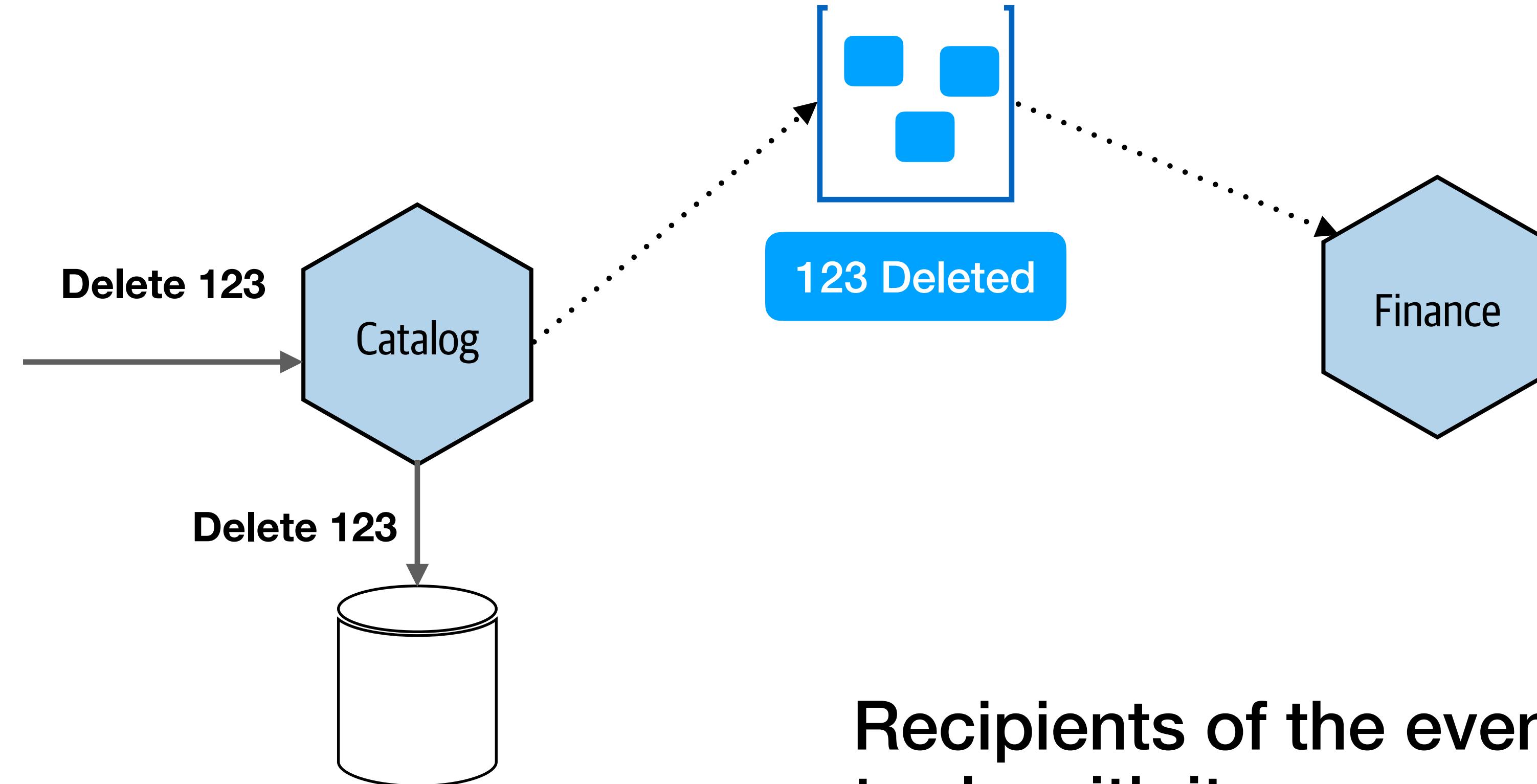
## CASCADING DELETES?



## CASCADING DELETES?

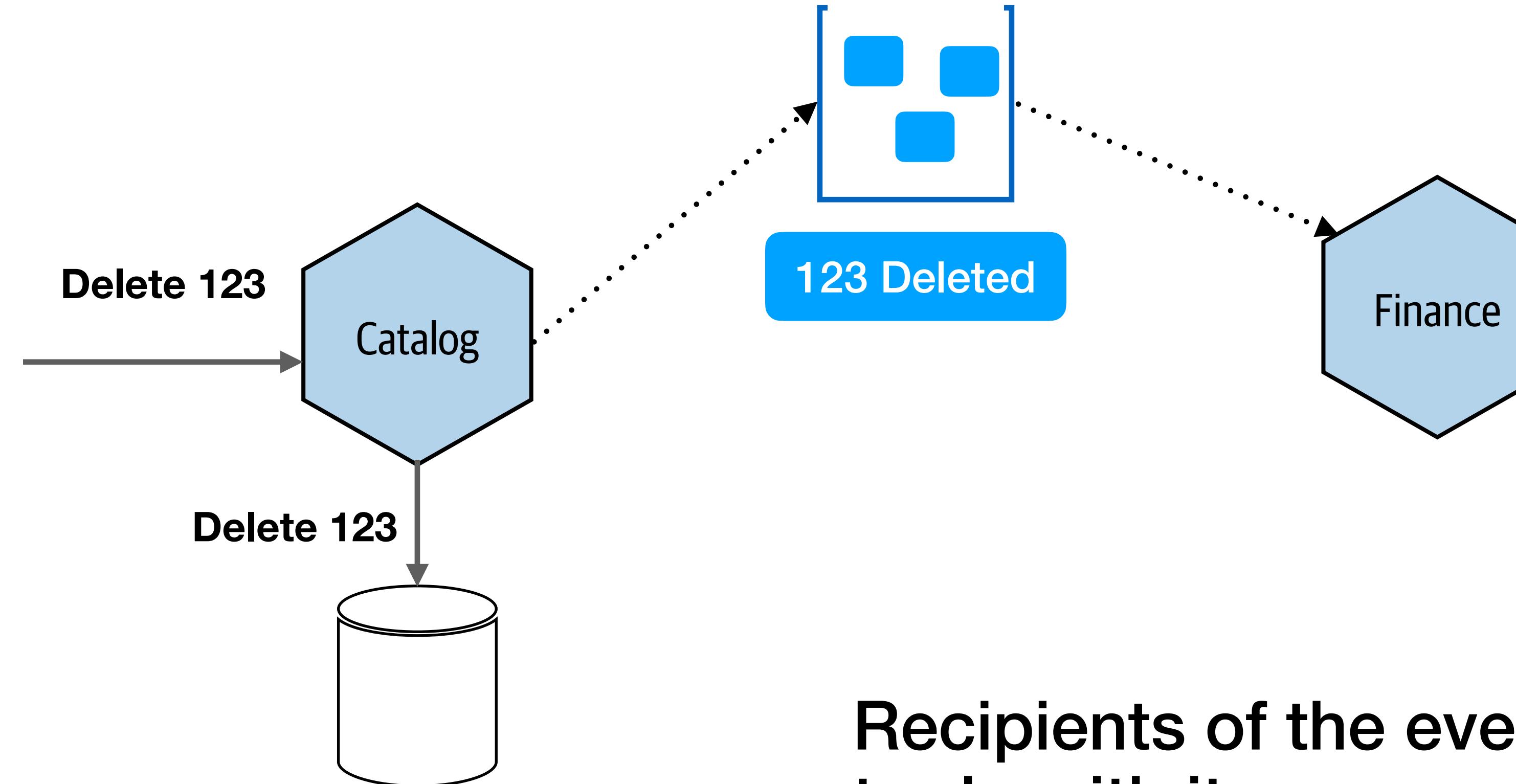


## CASCADING DELETES?



**Recipients of the event can decide  
to do with it...**

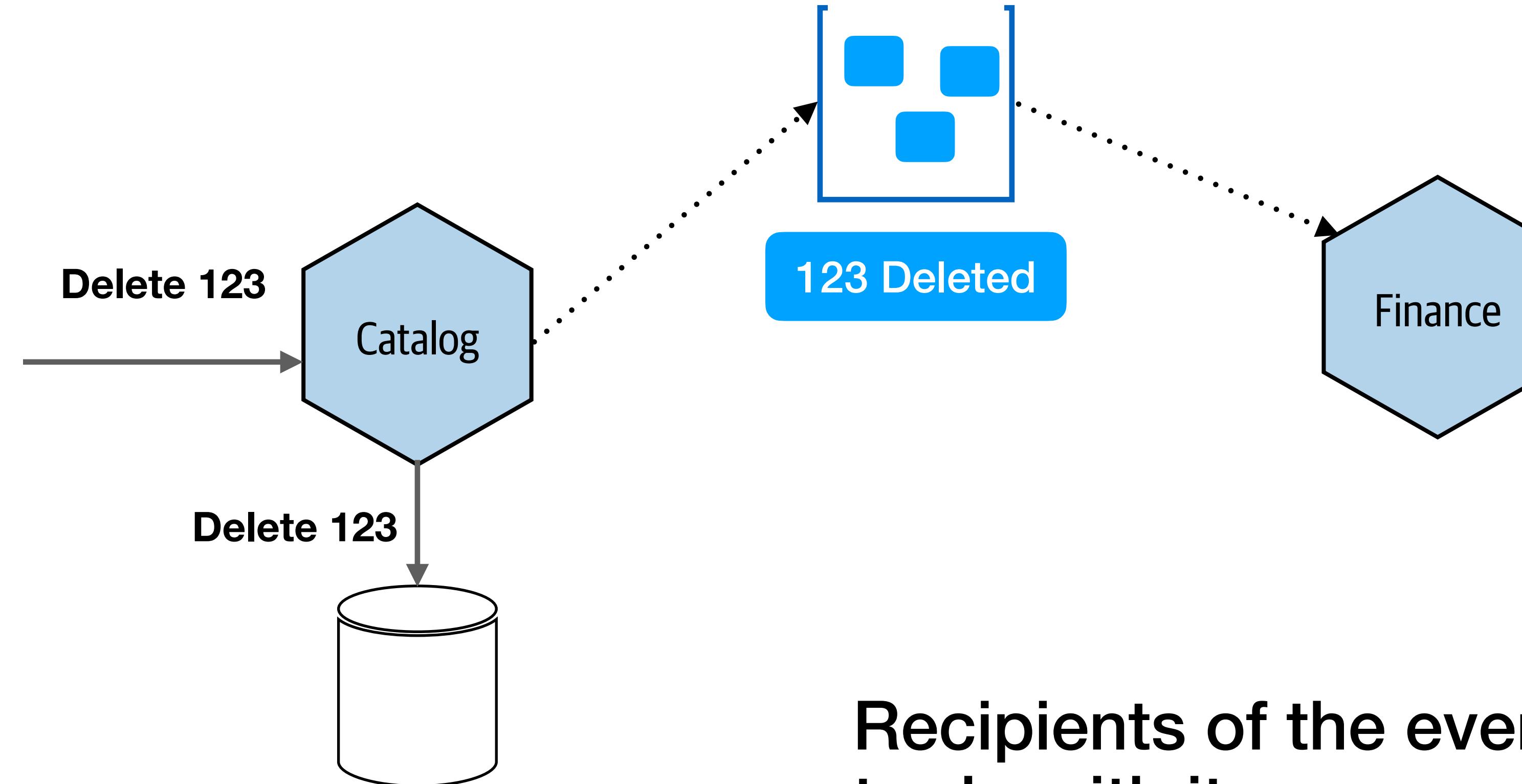
## CASCADING DELETES?



Recipients of the event can decide to do with it...

...you probably don't want to remove entries from the ledger!

## CASCADING DELETES?



Recipients of the event can decide to do with it...

...you probably don't want to remove entries from the ledger!

There is a delay between the event being fired and it being received

# EVENTUAL CONSISTENCY

## Eventually Consistent - Revisited

By Werner Vogels on 22 December 2008 04:15 PM | [Permalink](#) | [Comments \(\)](#)

*I wrote a [first version of this posting](#) on consistency models about a year ago, but I was never happy with it as it was written in haste and the topic is important enough to receive a more thorough treatment. [ACM Queue](#) asked me to revise it for use in their magazine and I took the opportunity to improve the article. This is that new version.*

### **Eventually Consistent - Building reliable distributed systems at a worldwide scale demands trade-offs between consistency and availability.**

At the foundation of Amazon's cloud computing are infrastructure services such as Amazon's S3 (Simple Storage Service), SimpleDB, and EC2 (Elastic Compute Cloud) that provide the resources for constructing Internet-scale computing platforms and a great variety of applications. The requirements placed on these infrastructure services are very strict; they need to score high marks in the areas of security, scalability, availability, performance, and cost effectiveness, and they need to meet these requirements while serving millions of customers around the globe, continuously.

Under the covers these services are massive distributed systems that operate on a worldwide scale. This scale creates additional challenges, because when a system processes trillions and trillions of requests, events that normally have a low probability of occurrence are now guaranteed to happen and need to be accounted for up front in the design and architecture of the system. Given the worldwide scope of these systems, we use replication techniques ubiquitously to guarantee consistent performance and high availability. Although replication brings us closer to our goals, it cannot achieve them in a perfectly transparent manner; under a number of conditions the customers of these services will be confronted with the consequences of using replication techniques inside the services.



#### Contact Info

**Werner Vogels**  
CTO - [Amazon.com](#)

[werner@allthingsdistributed.com](mailto:werner@allthingsdistributed.com)

#### Other places

Follow werner on [twitter](#) if you want to know what he is current reading or thinking about.  
At [werner.ly](http://werner.ly) he posts material that doesn't belong on this blog or on [twitter](#).

[https://www.allthingsdistributed.com/2008/12/eventually\\_consistent.html](https://www.allthingsdistributed.com/2008/12/eventually_consistent.html)

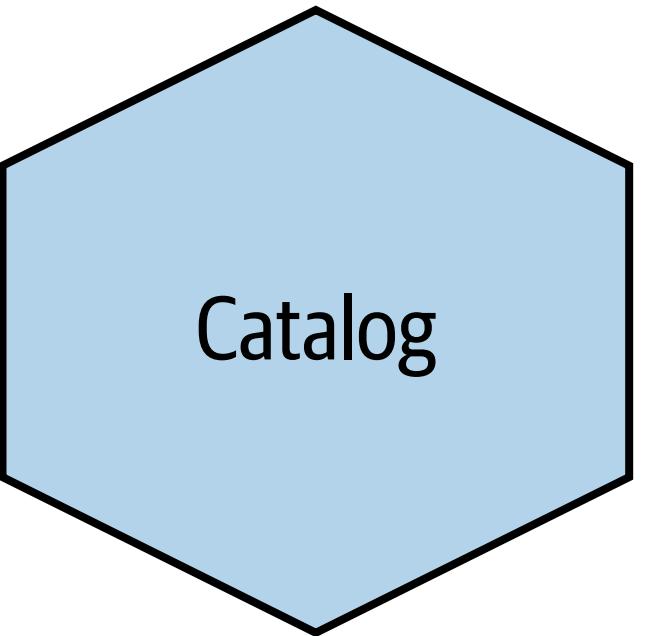
## EVENTUAL CONSISTENCY DEFINED...

***“A system is convergent or ‘eventually consistent’ if, when all messages have been delivered, all replicas agree on the set of stored values.”***

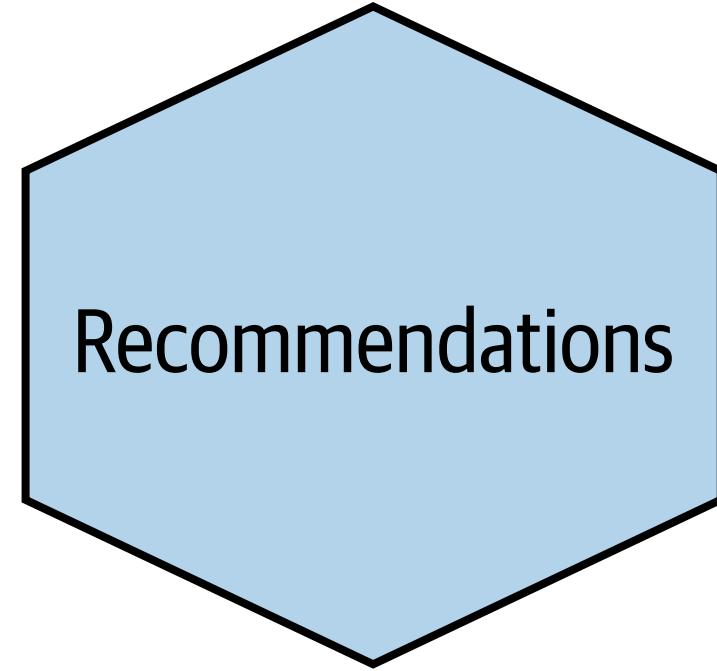
- Peter Alvaro

<https://www2.eecs.berkeley.edu/Pubs/TechRpts/2015/EECS-2015-242.pdf>

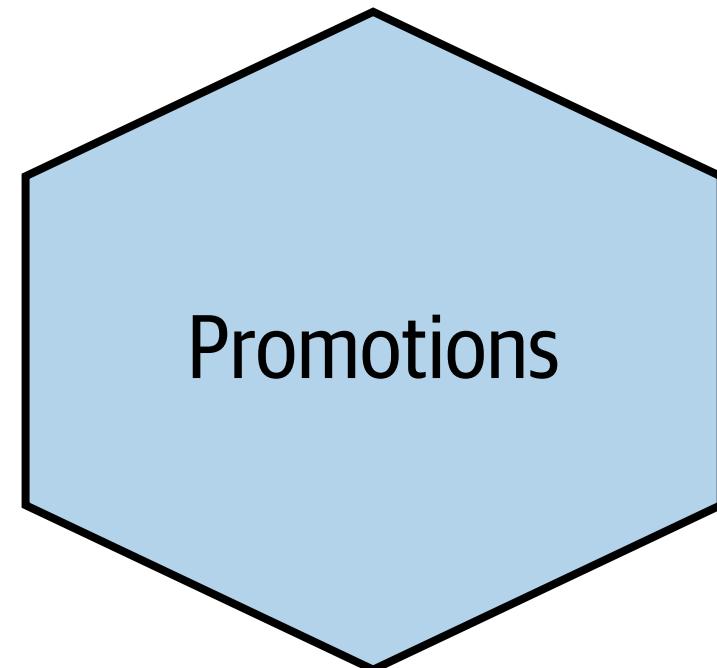
## EXAMPLE...



**Current Price: €4.99**

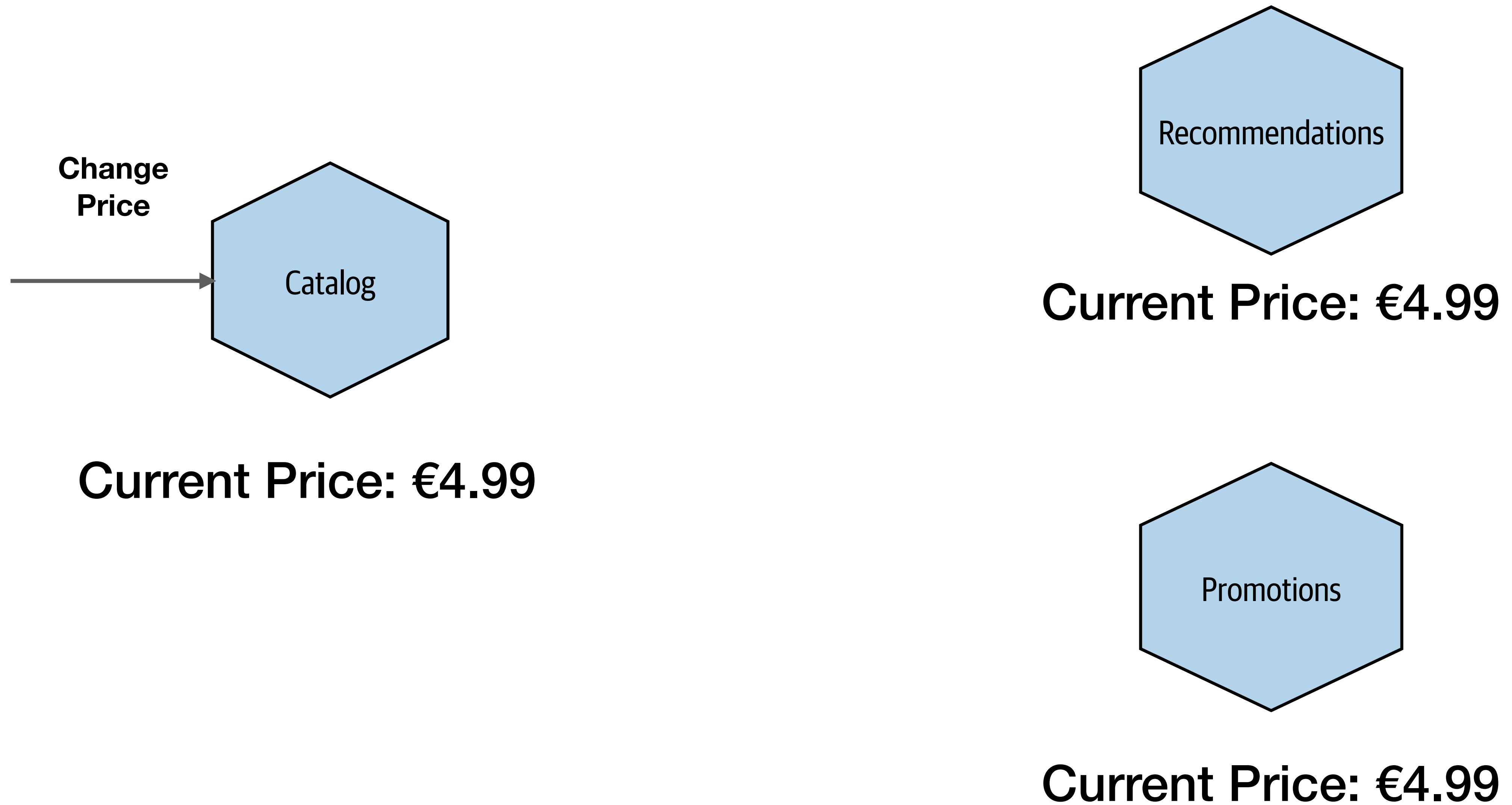


**Current Price: €4.99**

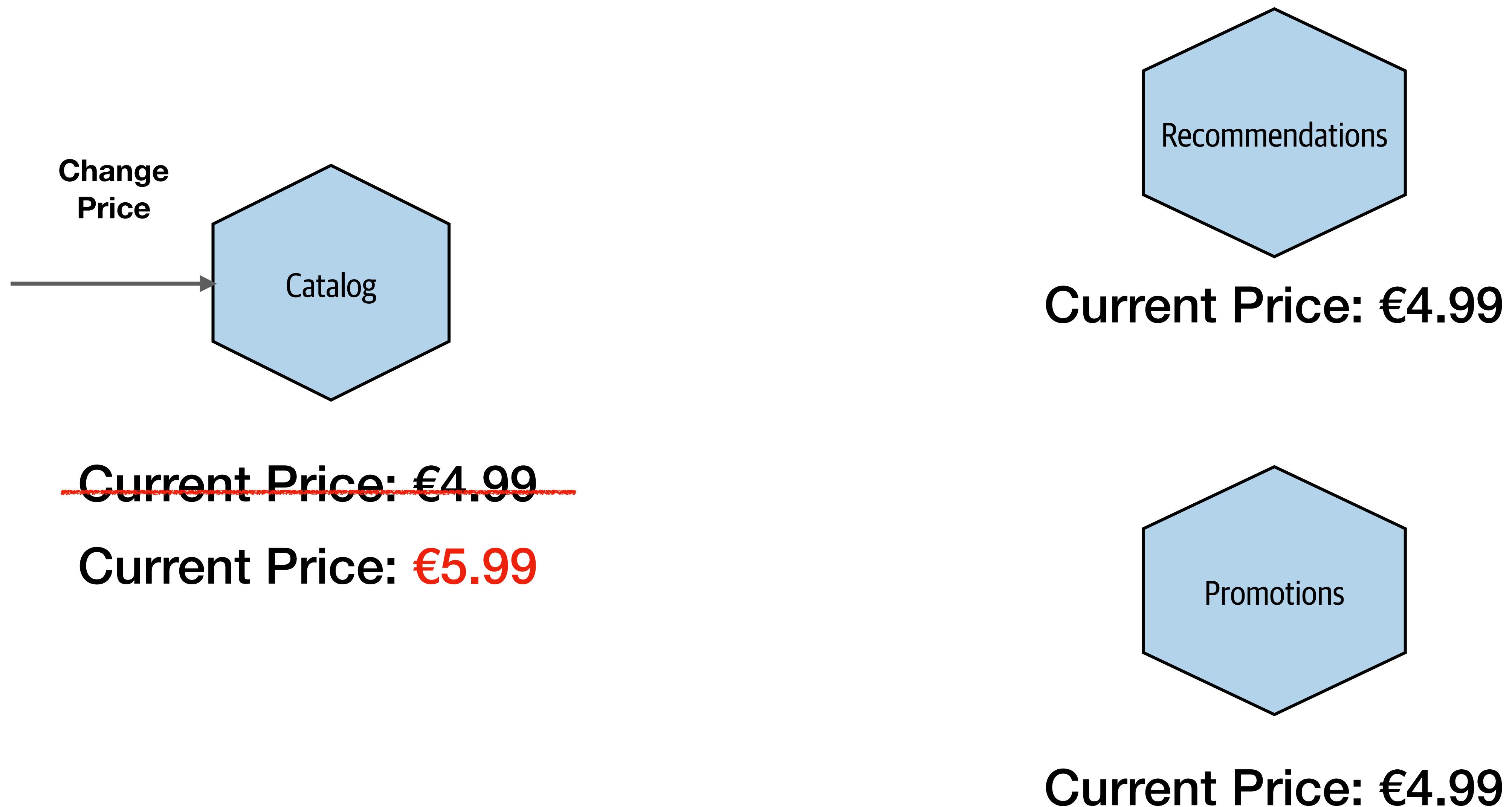


**Current Price: €4.99**

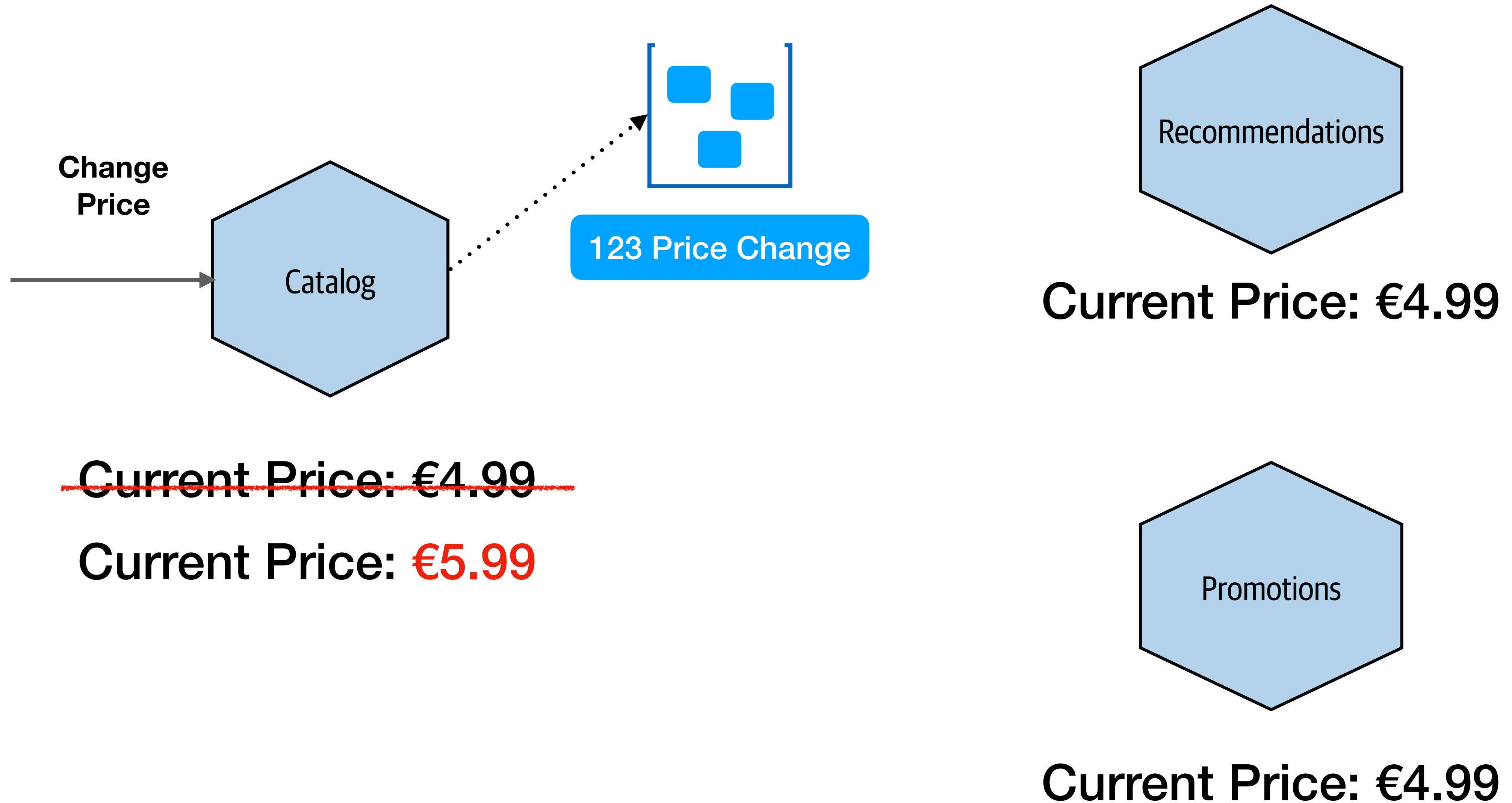
## EXAMPLE...



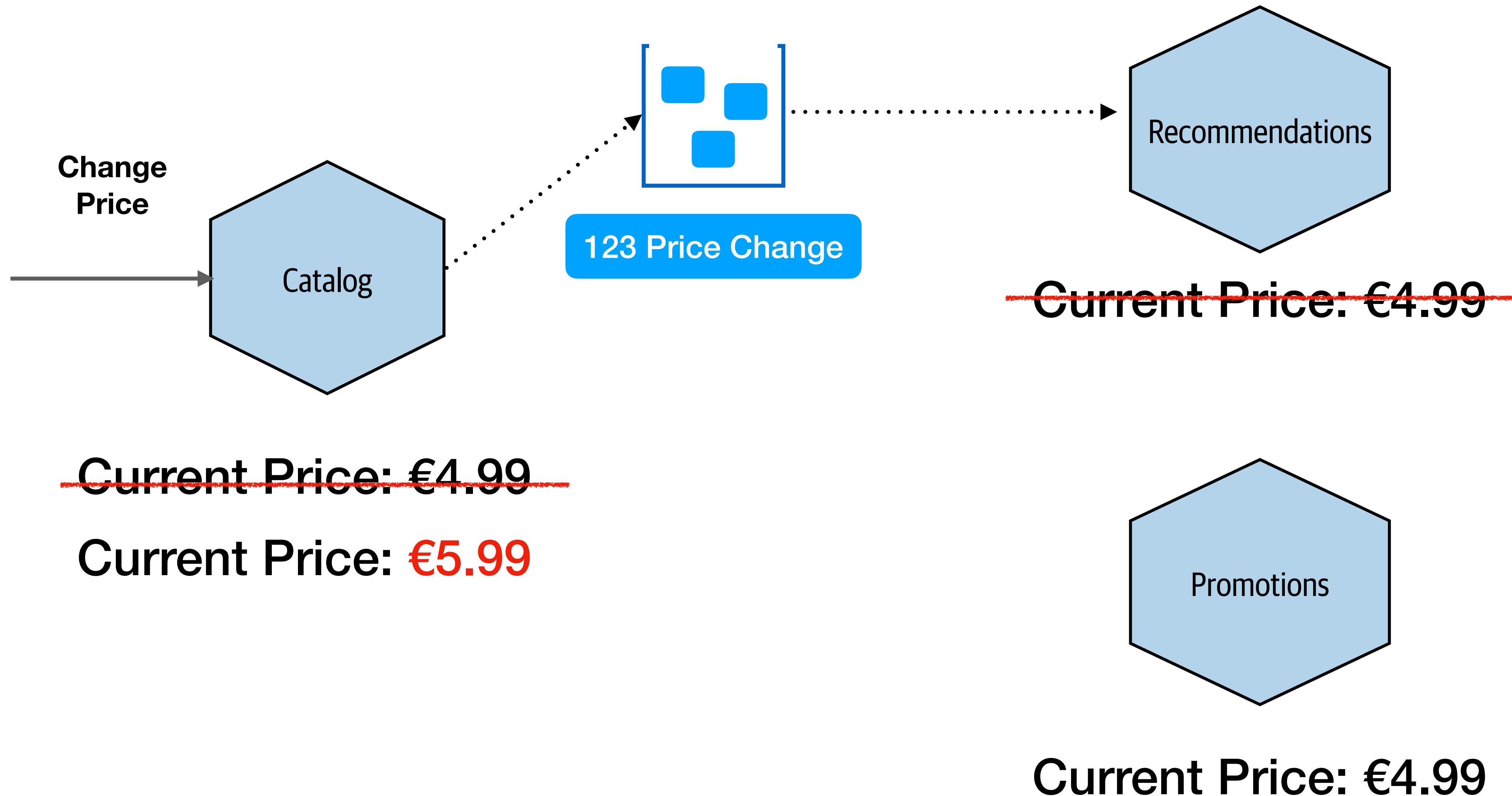
## EXAMPLE...



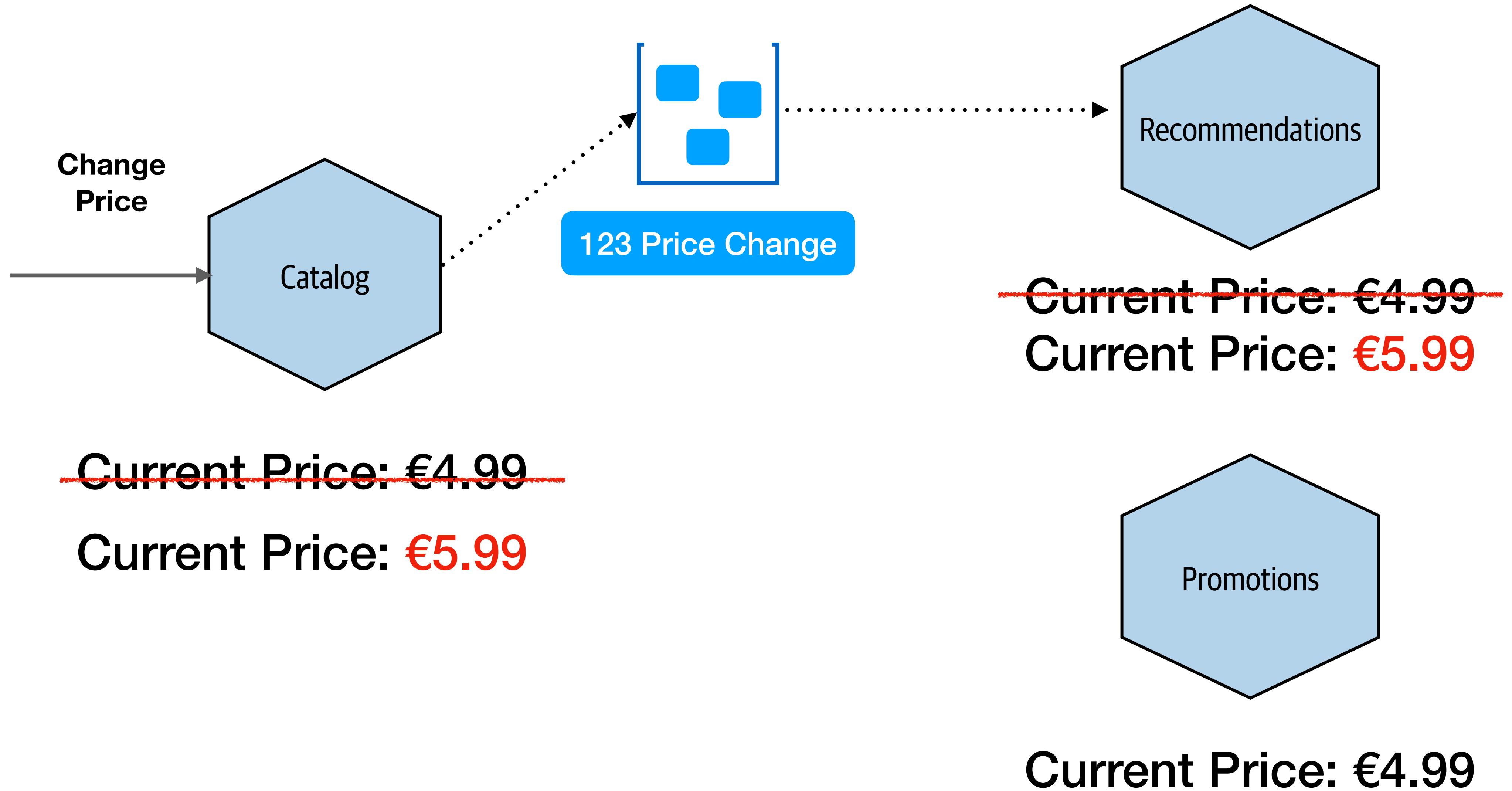
## EXAMPLE...



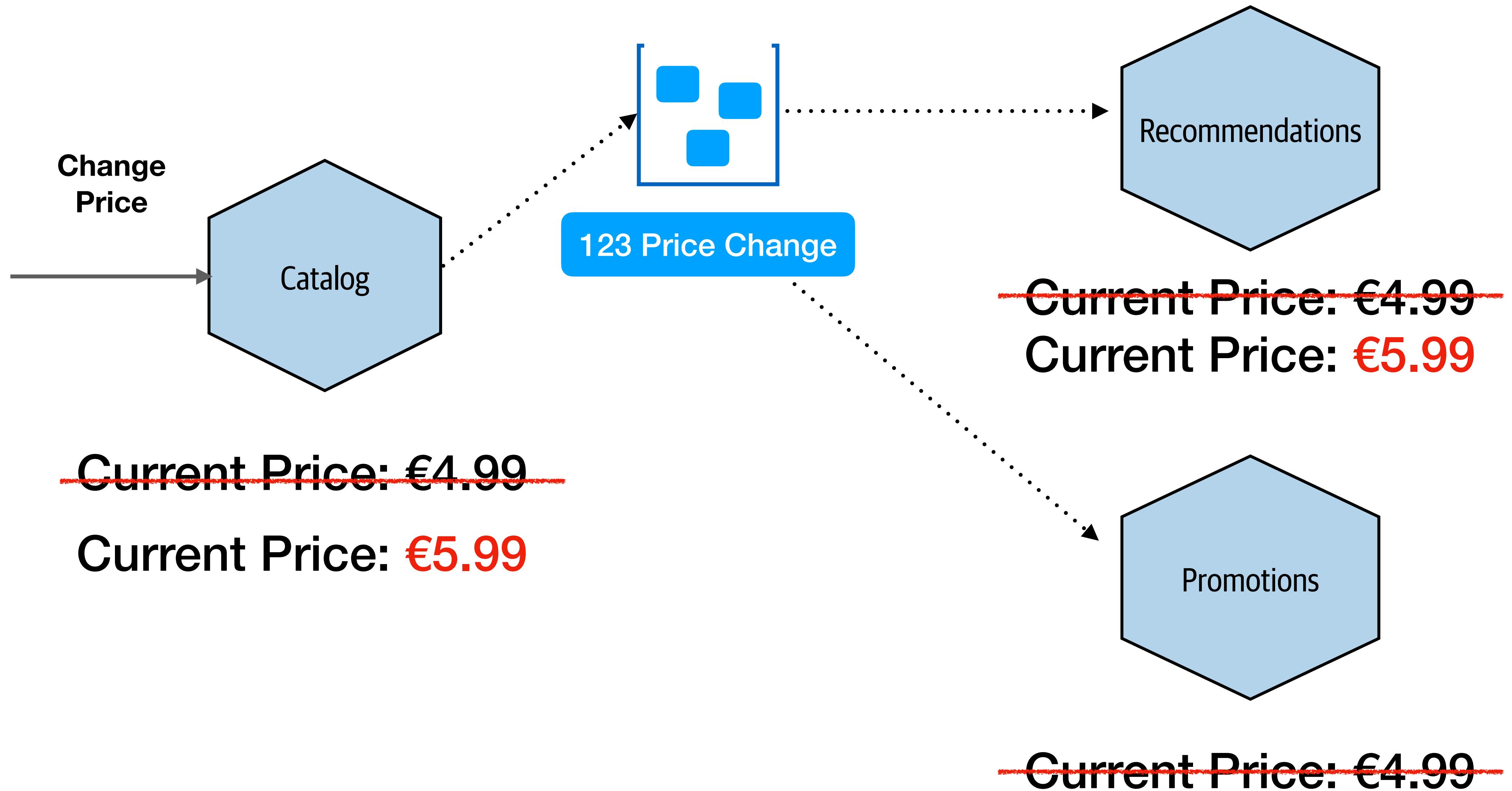
## EXAMPLE...



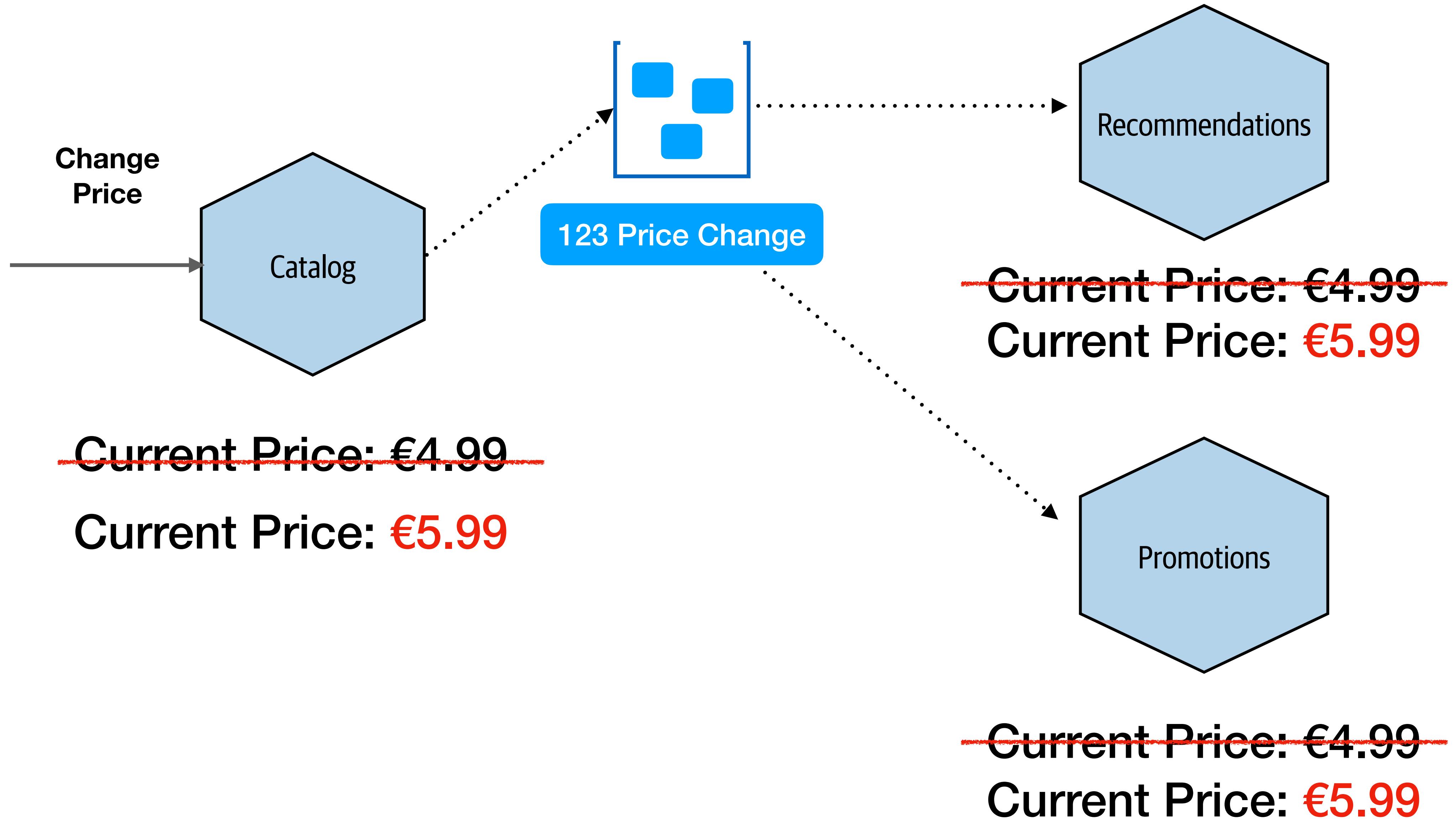
## EXAMPLE...



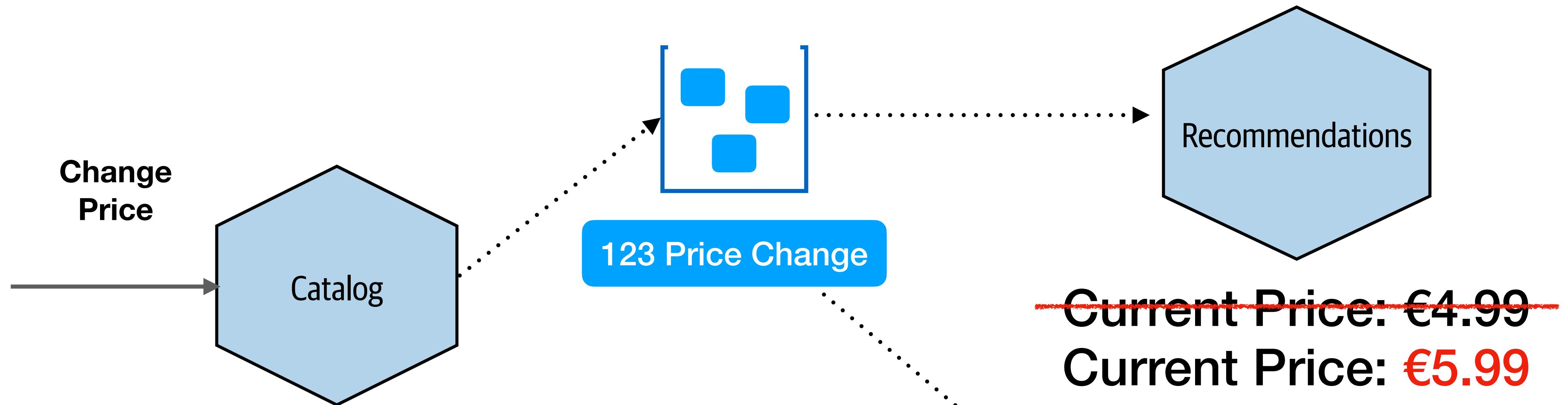
## EXAMPLE...



## EXAMPLE...



## EXAMPLE...



~~Current Price: €4.99~~

Current Price: **€5.99**

Once all the messages are processed by the replicas, they agree!

~~Current Price: €4.99~~

Current Price: **€5.99**

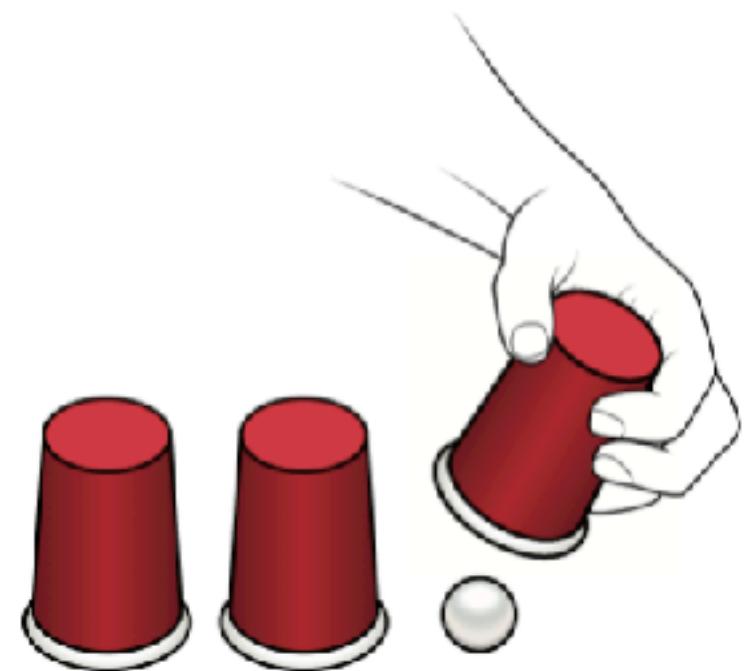
# OR EVENTUALLY CONVERGENT?

## Don't Get Stuck in the CON Game (V3)

Consistency, convergence, and confluence are not the same! Eventual consistency and eventual convergence aren't the same as confluence, either.



Pat Helland  
Jul 1



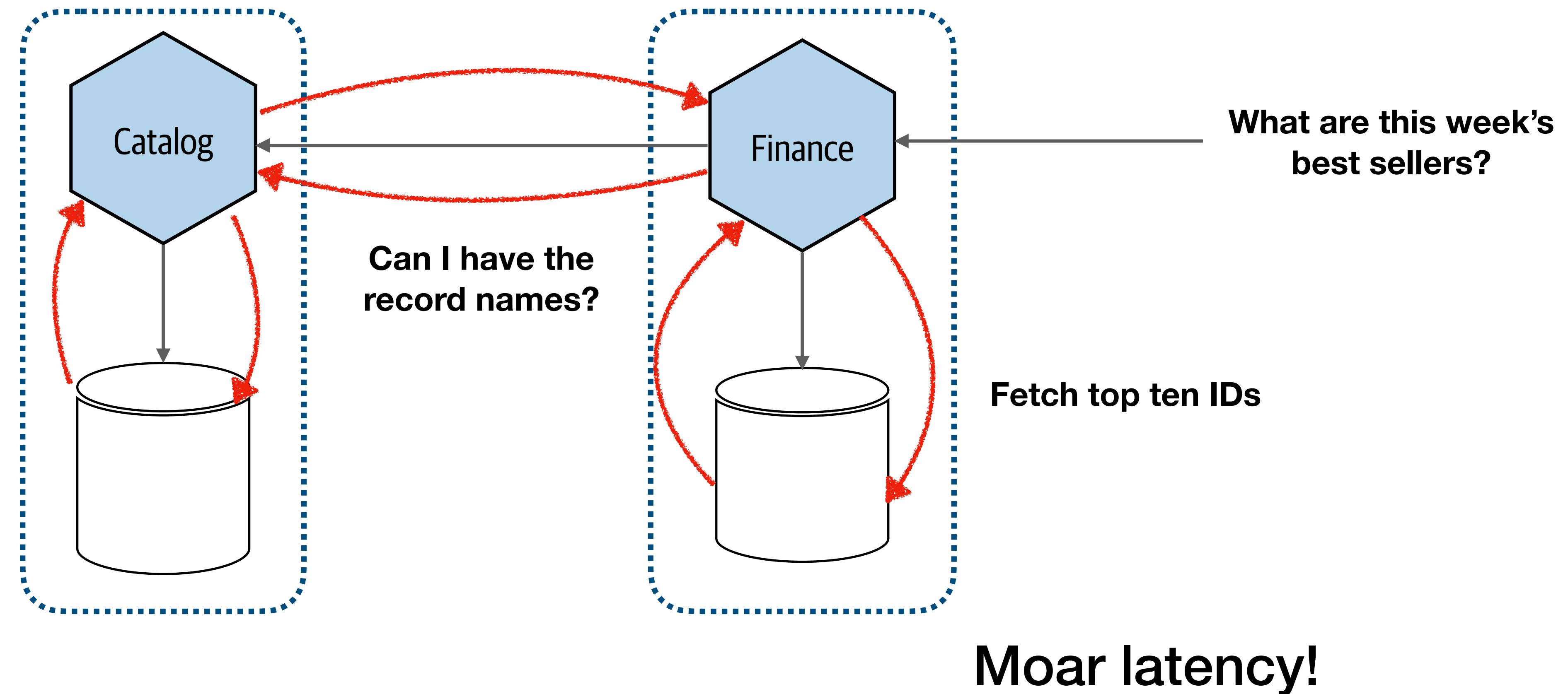
This is the 3rd version of “Don’t Get Stuck in the CON Game” that I’ve blogged.

Like many others, I’ve fallen victim to using the phrase *eventual consistency*. It’s a popular phrase, even though its meaning is fuzzy. Different communities within computer science use the word *consistency* in varying ways. Even within these different communities, people are inconsistent in their use of *consistency*. That fuzziness has gotten many of us tangled up in misunderstanding.

It turns out that there are other terms, *convergence* and *confluence*, that have crisper definitions and are more easily understood than *consistency*.

<https://pathelland.substack.com/p/dont-get-stuck-in-the-con-game-v3>

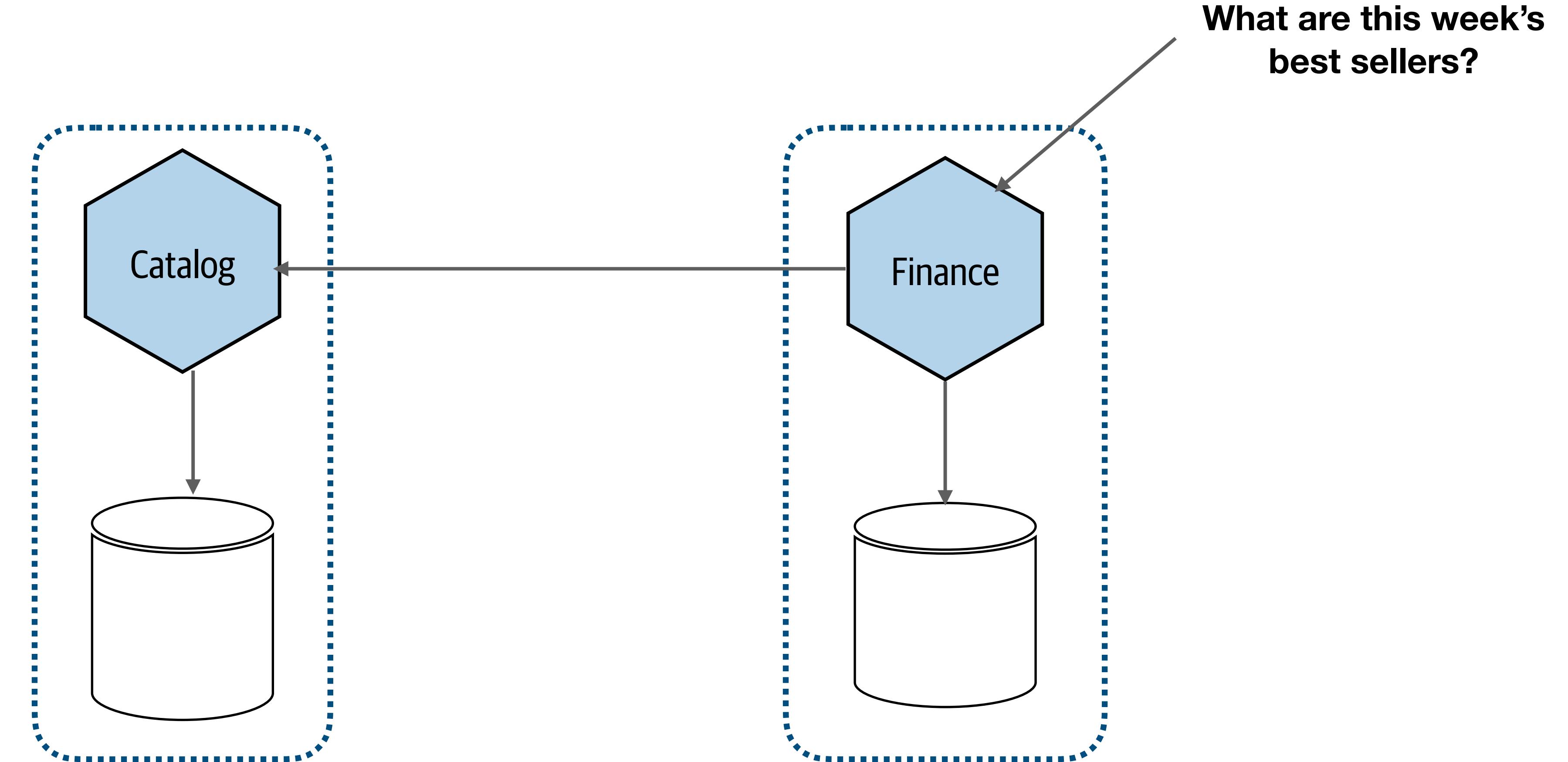
## JOINS AT THE SERVICES TIER



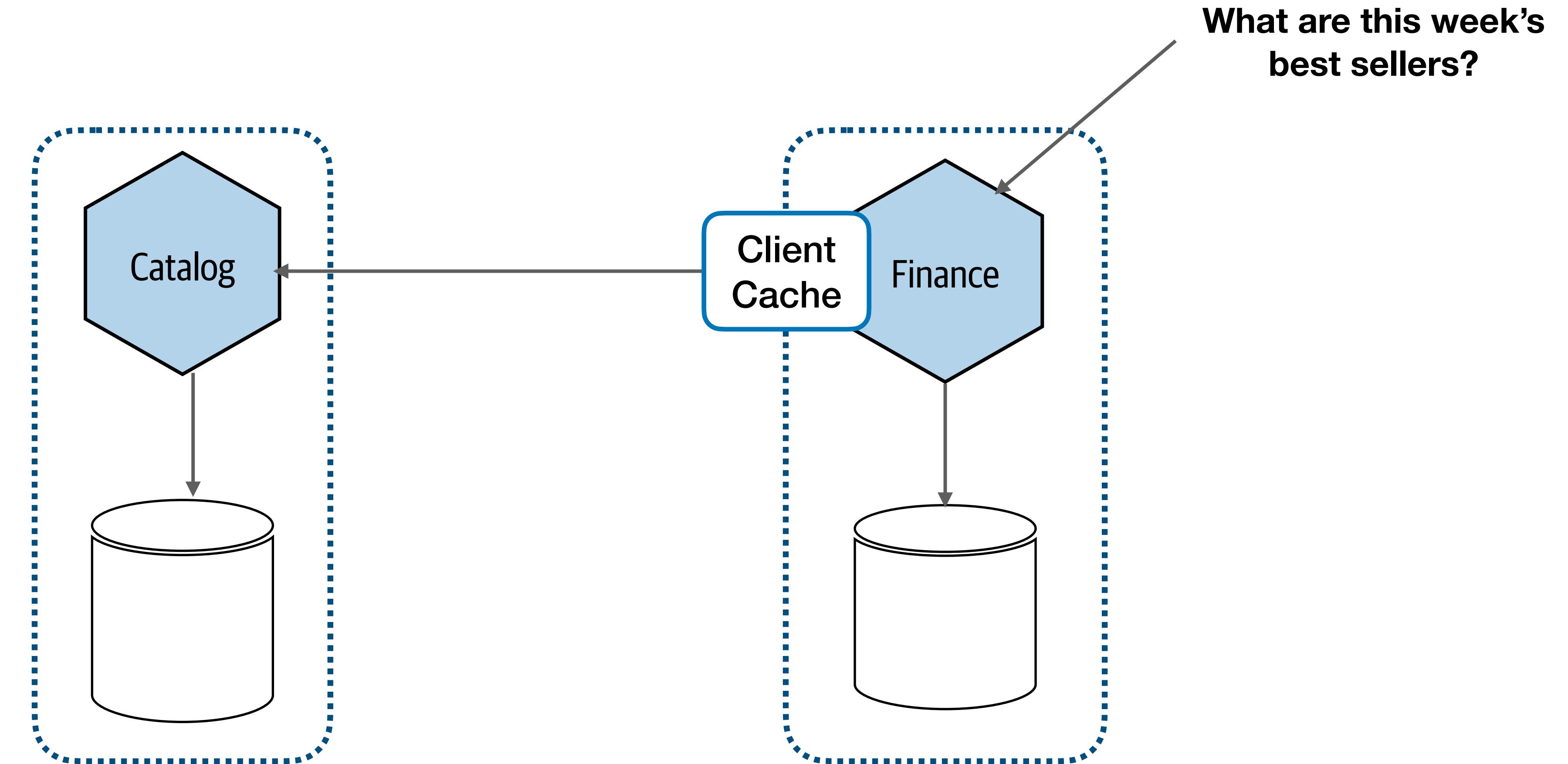
**How can we make this join more  
efficient?**

# Caching!

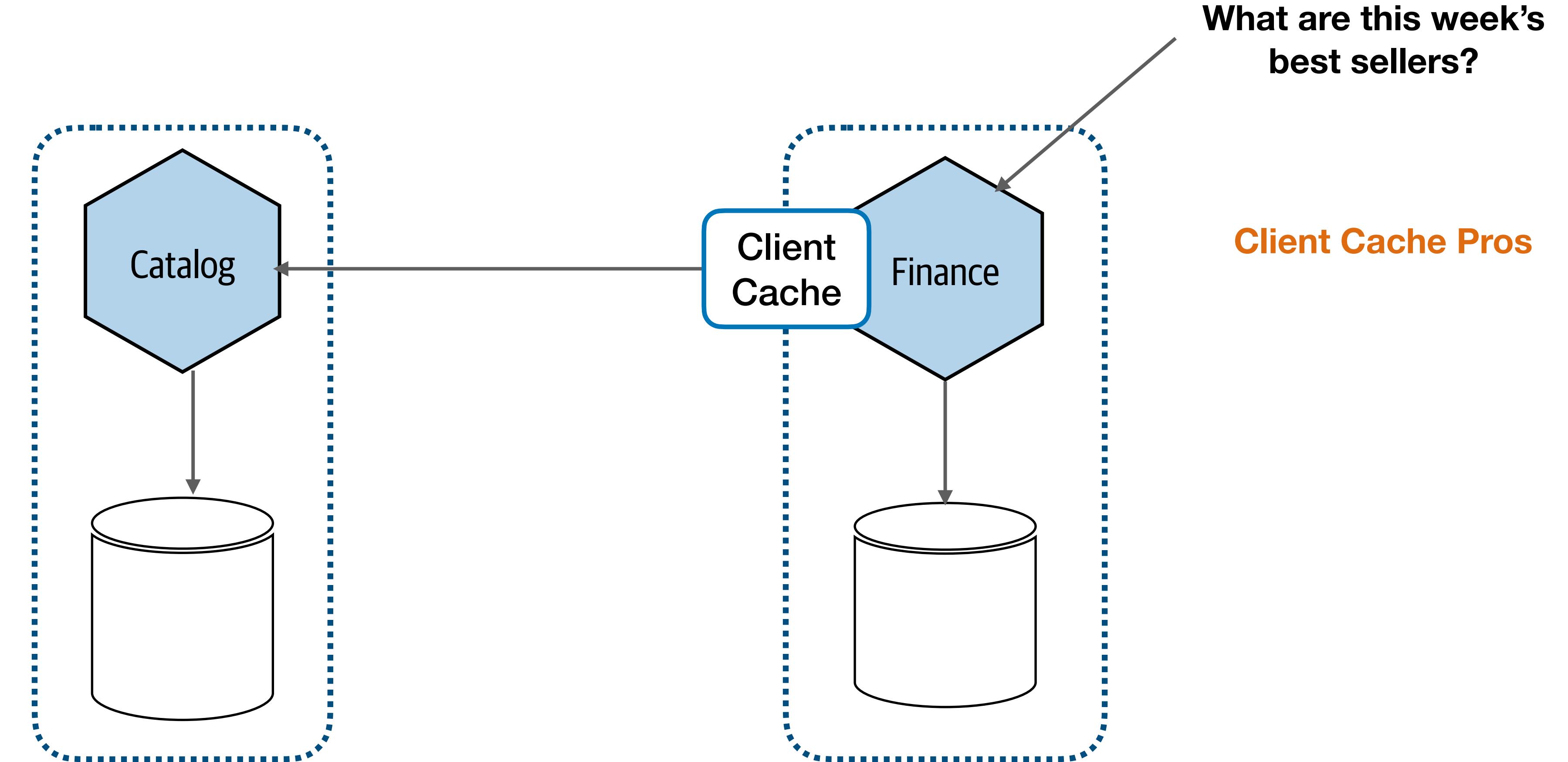
## WHERE COULD WE CACHE?



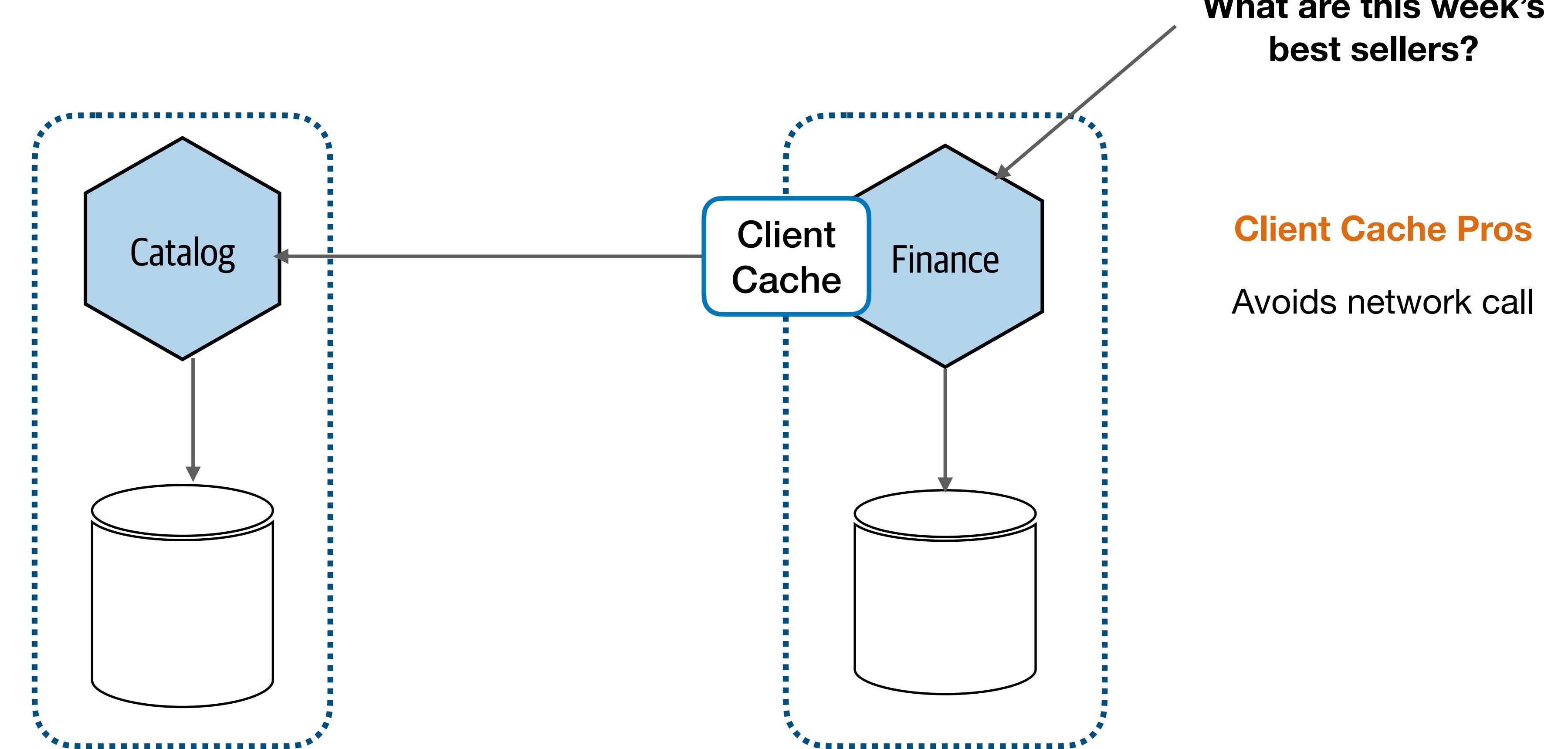
## WHERE COULD WE CACHE?



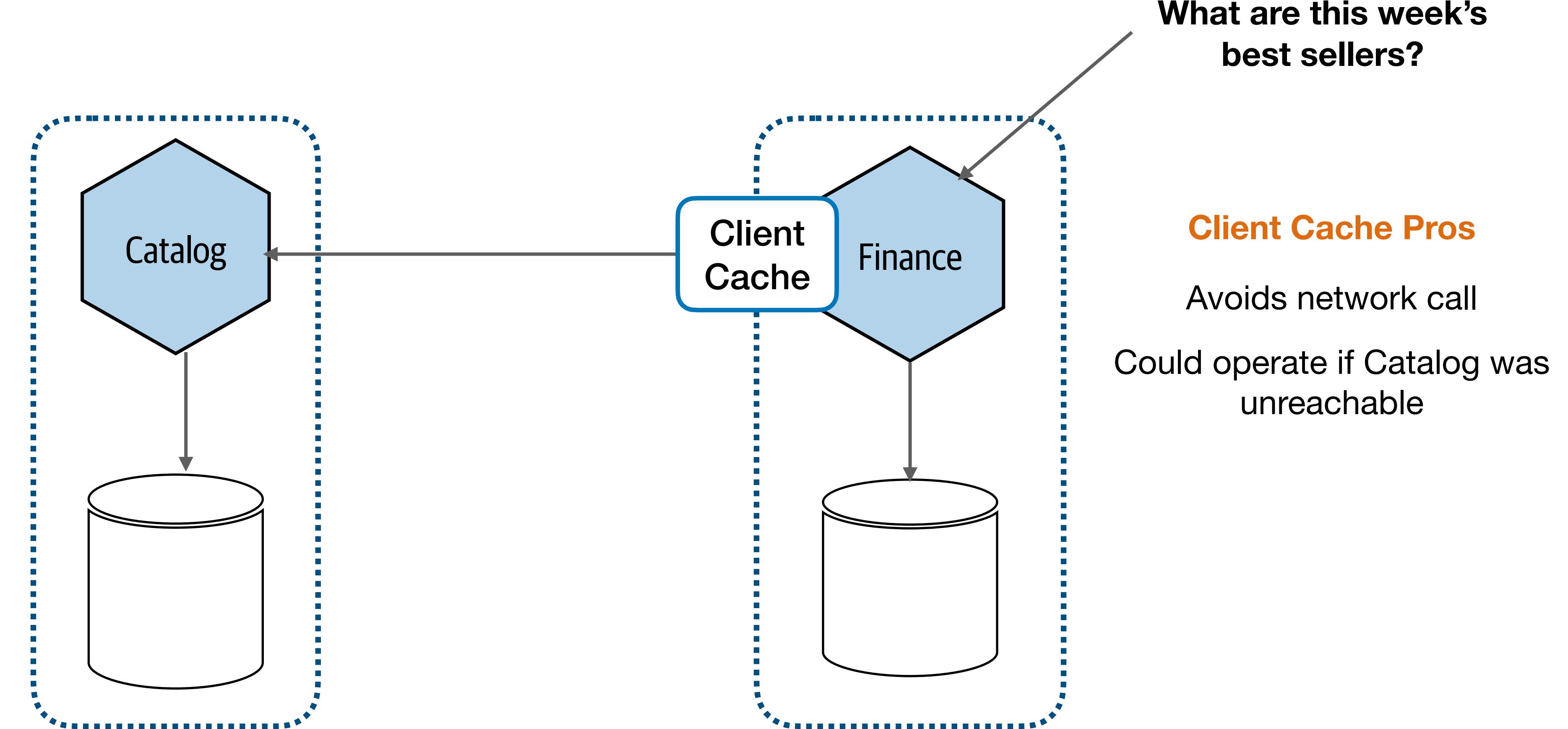
# WHERE COULD WE CACHE?



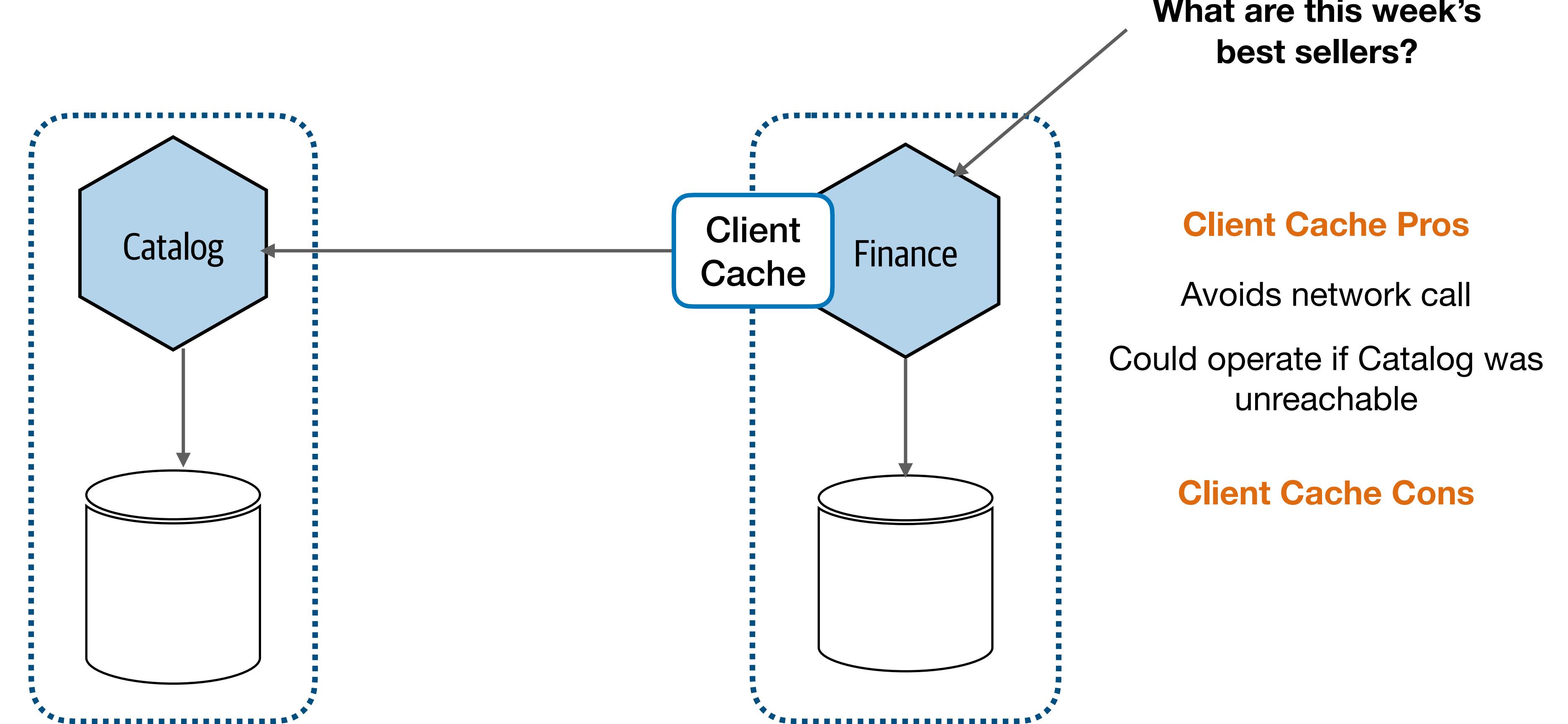
## WHERE COULD WE CACHE?



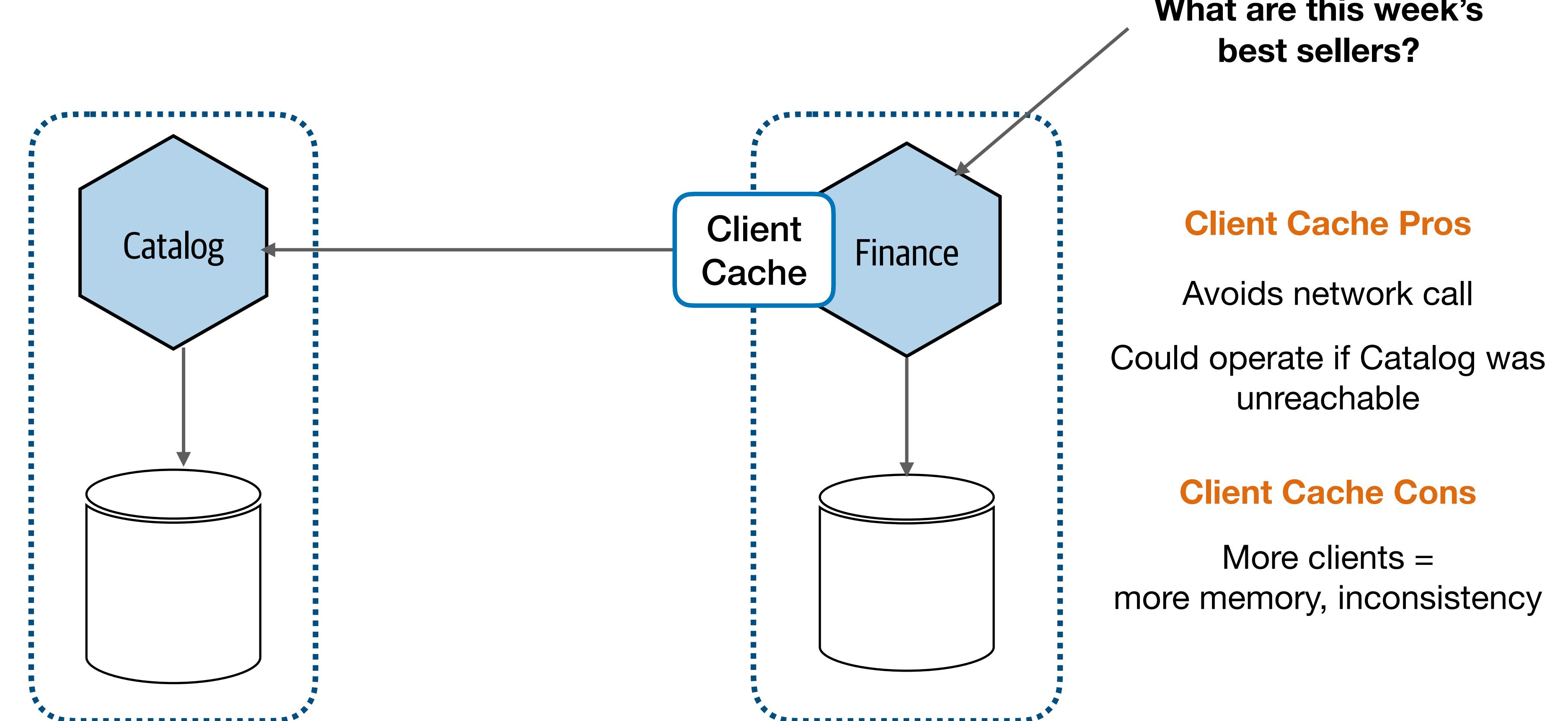
# WHERE COULD WE CACHE?



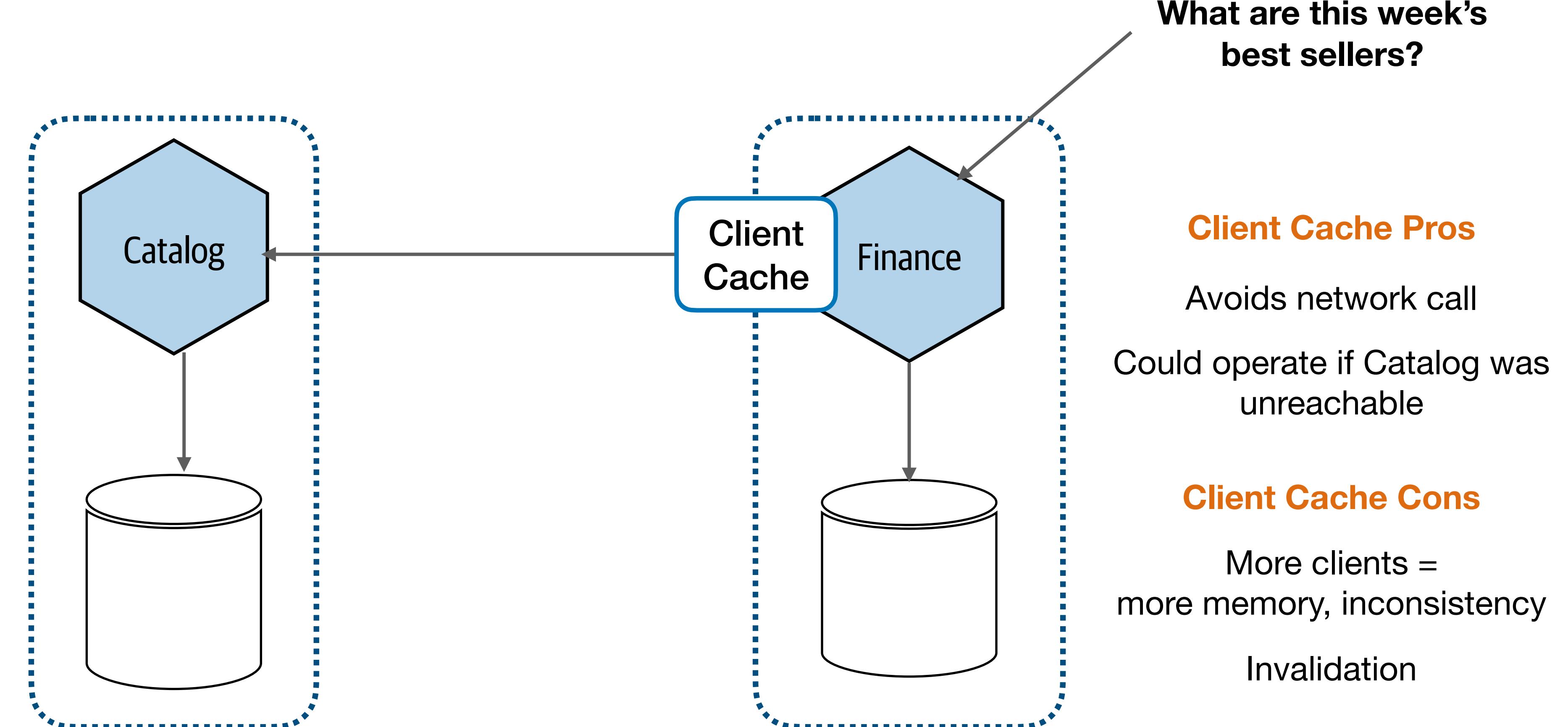
# WHERE COULD WE CACHE?



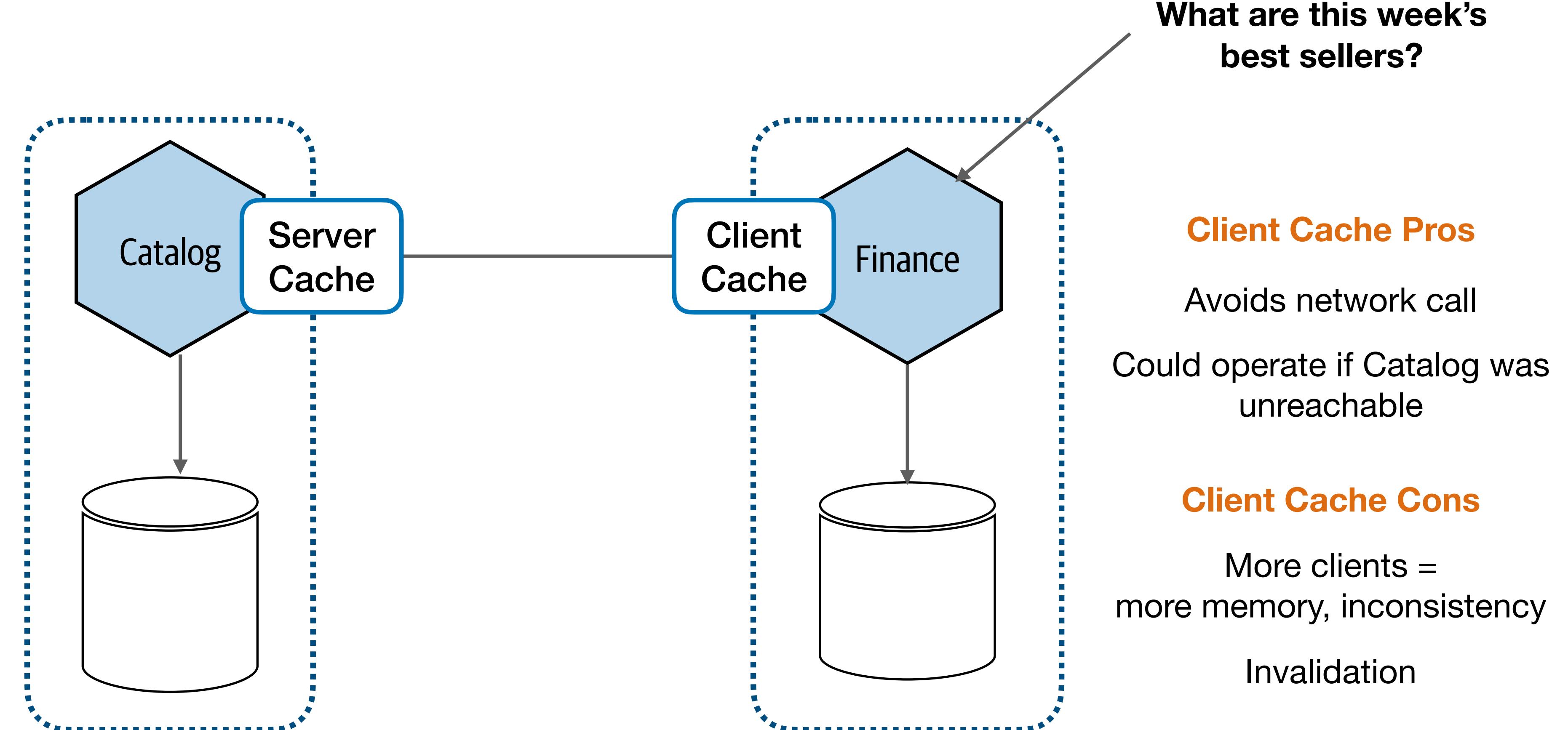
# WHERE COULD WE CACHE?



# WHERE COULD WE CACHE?

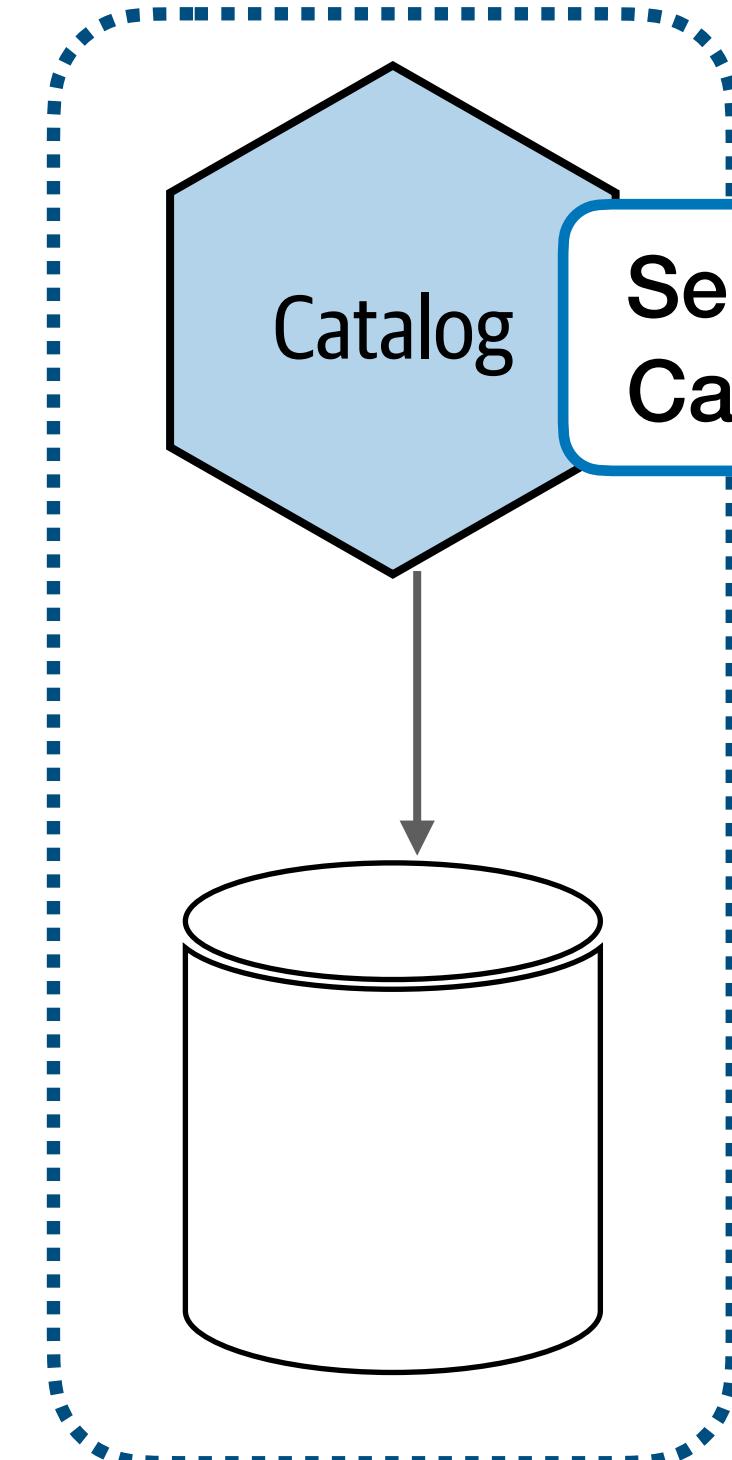


# WHERE COULD WE CACHE?

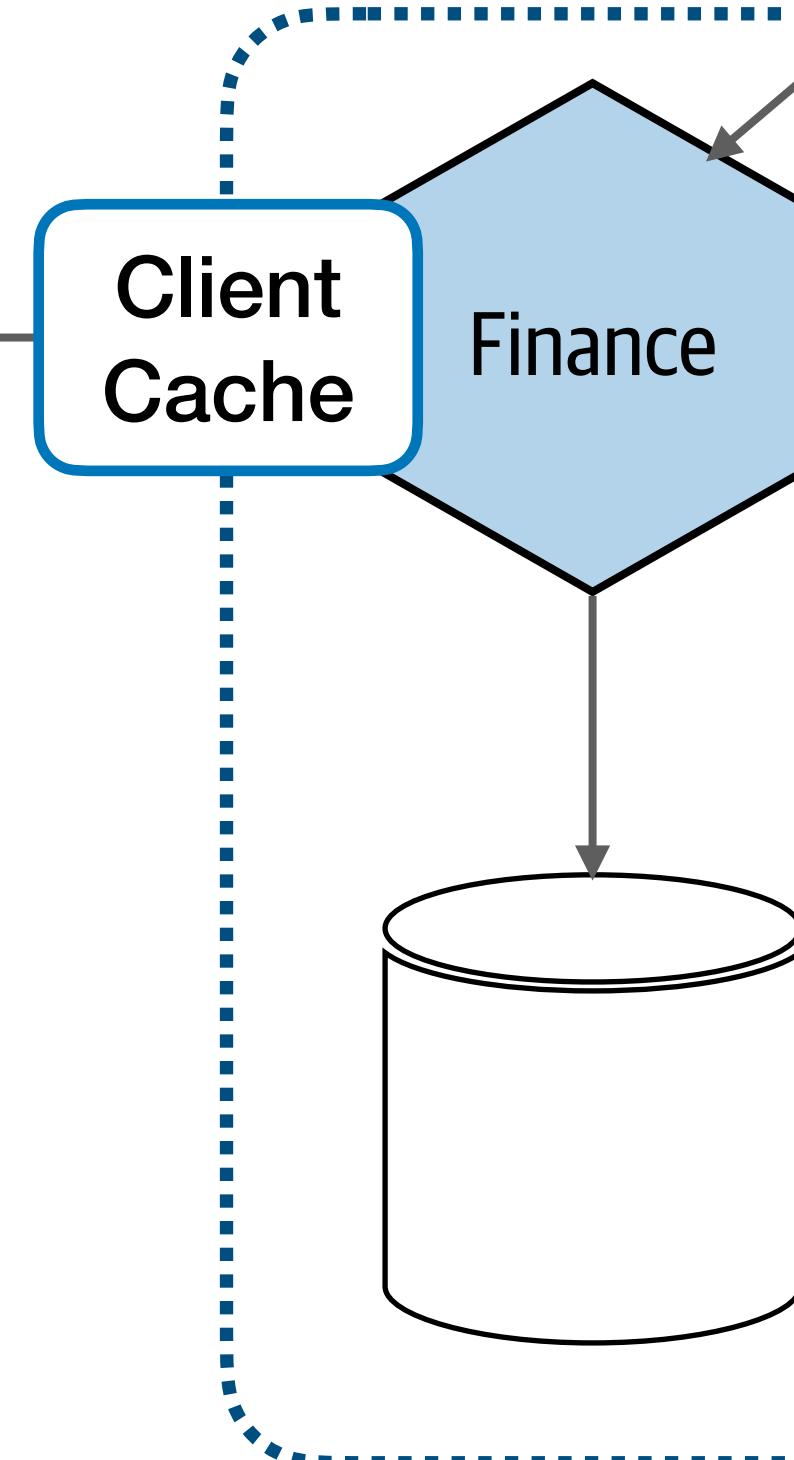


# WHERE COULD WE CACHE?

## Server Cache Pros



What are this week's best sellers?



## Client Cache Pros

Avoids network call

Could operate if Catalog was unreachable

## Client Cache Cons

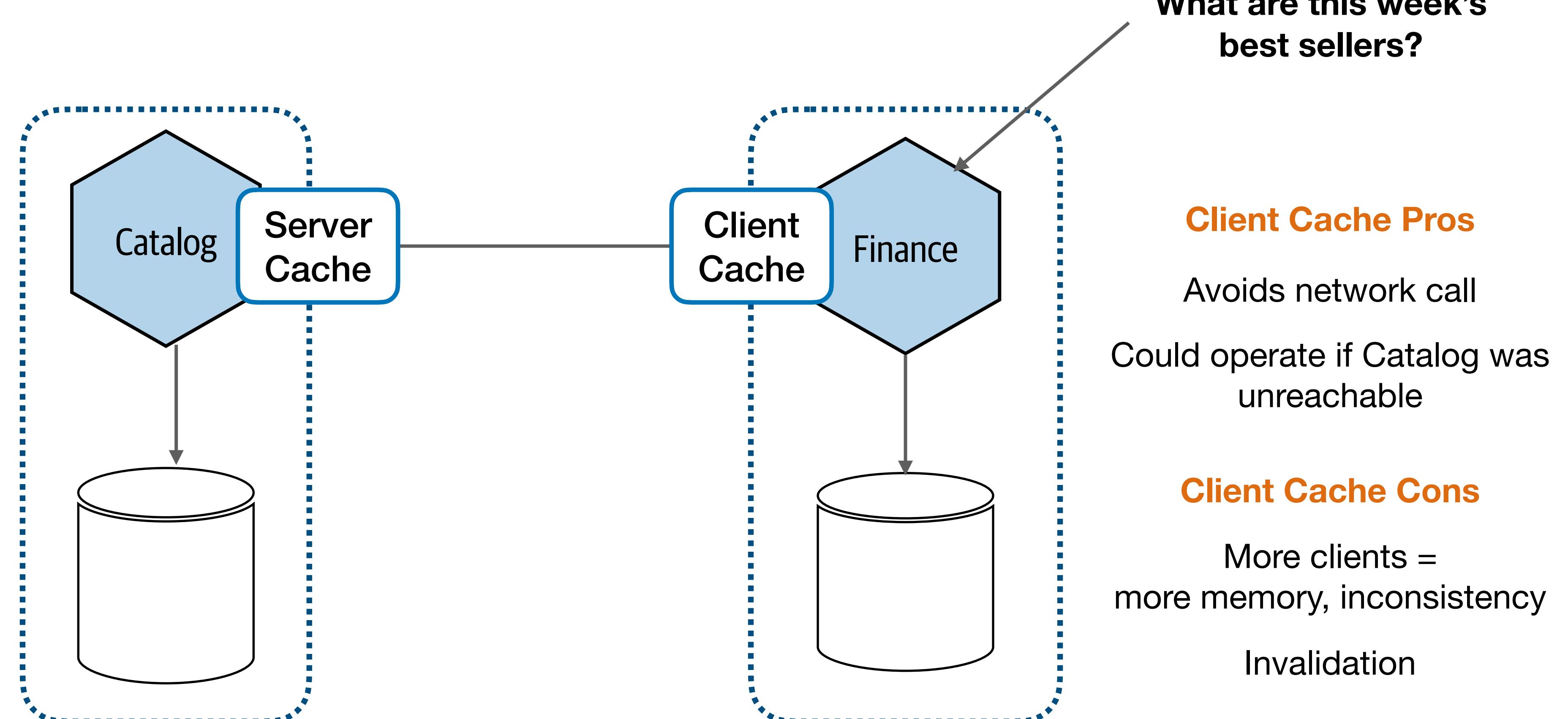
More clients = more memory, inconsistency

Invalidation

# WHERE COULD WE CACHE?

## Server Cache Pros

Invalidation easier



## Client Cache Pros

Avoids network call

Could operate if Catalog was unreachable

## Client Cache Cons

More clients =  
more memory, inconsistency

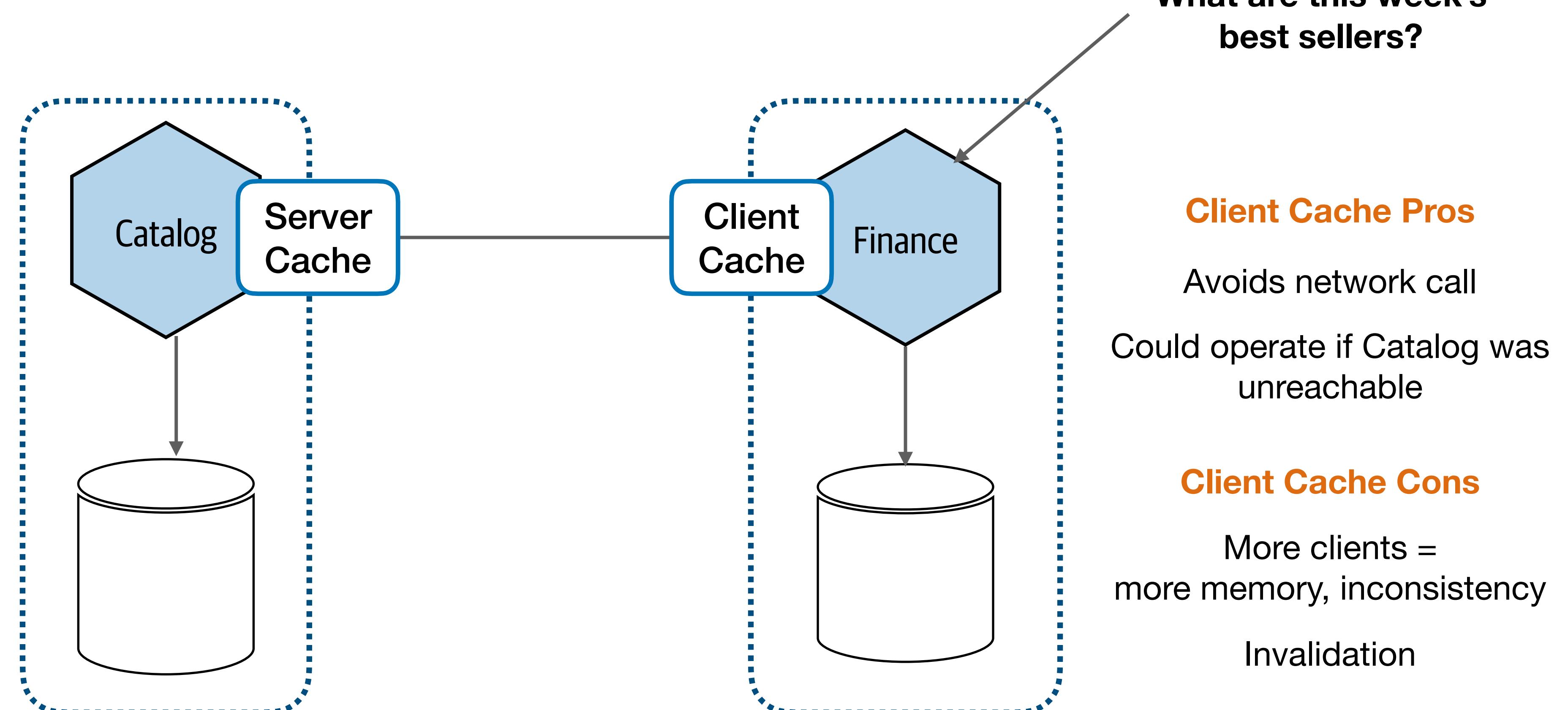
Invalidation

# WHERE COULD WE CACHE?

## Server Cache Pros

Invalidation easier

Can be more efficient in terms of memory



## Client Cache Pros

Avoids network call

Could operate if Catalog was unreachable

## Client Cache Cons

More clients = more memory, inconsistency

Invalidation

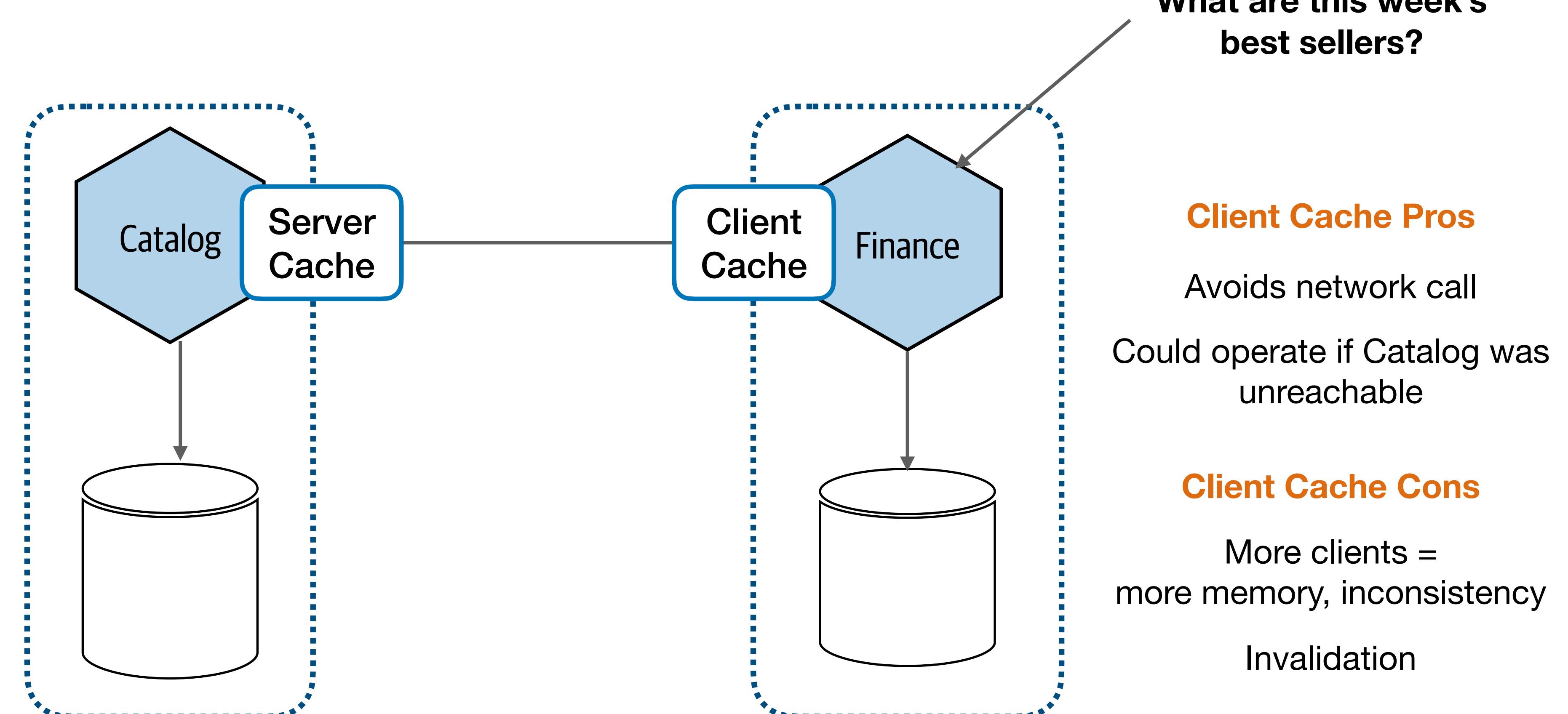
# WHERE COULD WE CACHE?

## Server Cache Pros

Invalidation easier

Can be more efficient in terms of memory

## Server Cache Cons



# WHERE COULD WE CACHE?

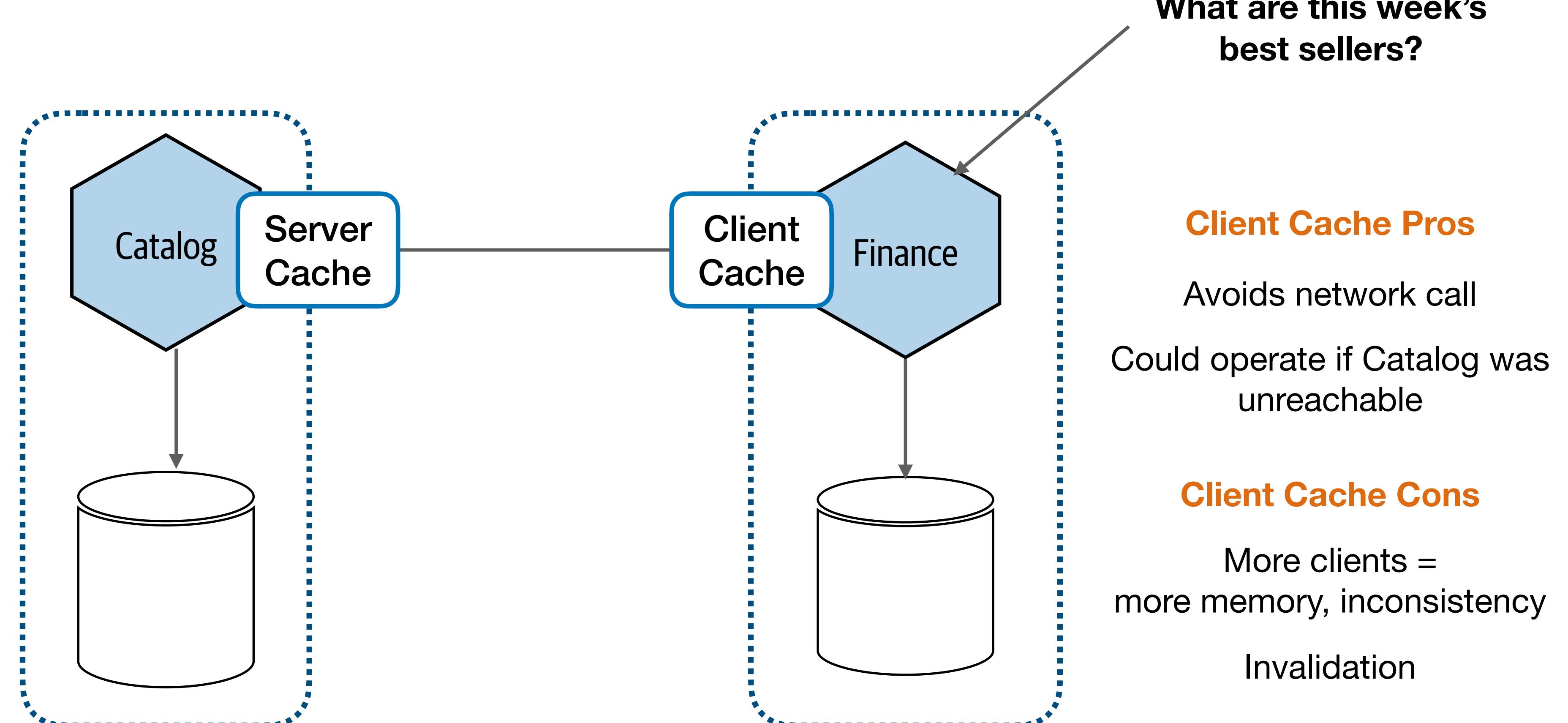
## Server Cache Pros

Invalidation easier

Can be more efficient in terms of memory

## Server Cache Cons

Call still needs to be made



# WHERE COULD WE CACHE?

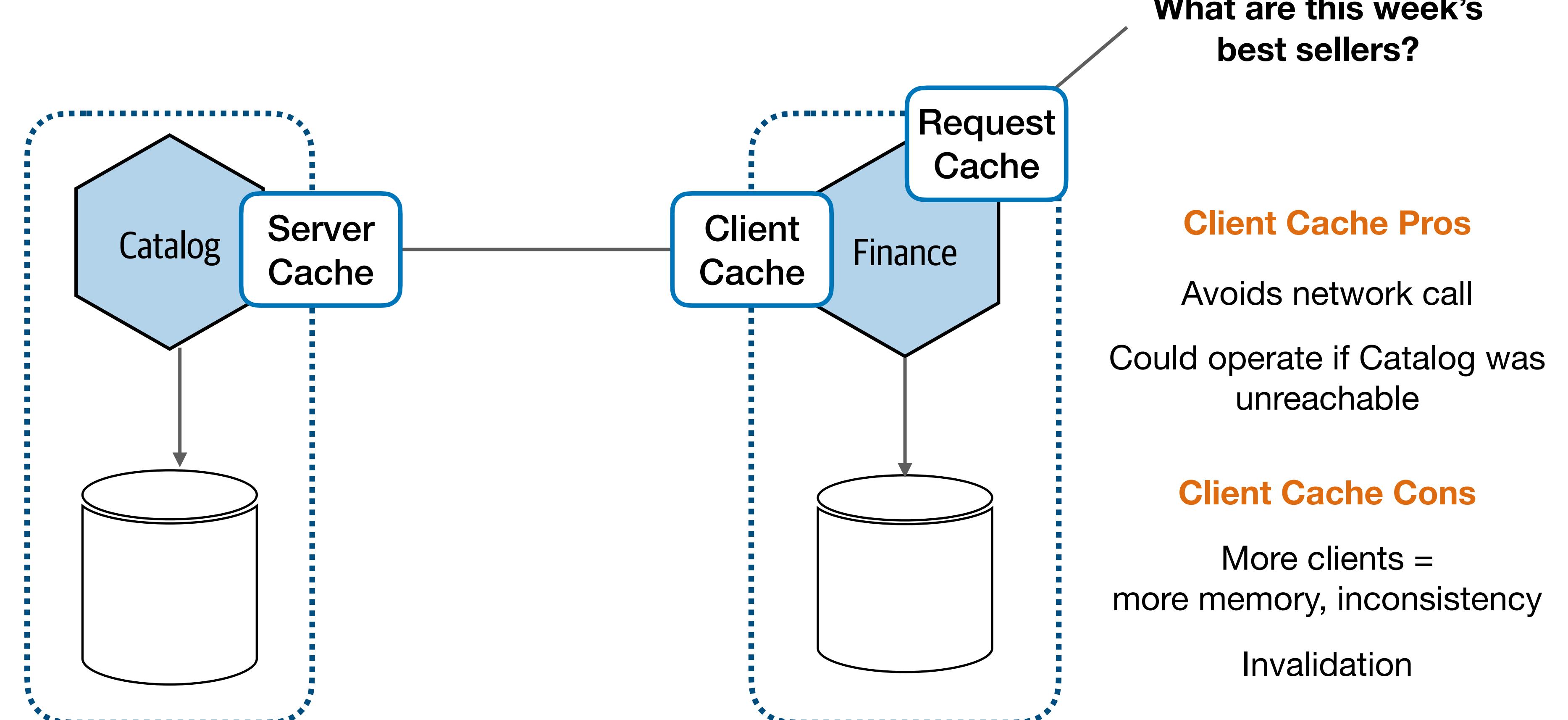
## Server Cache Pros

Invalidation easier

Can be more efficient in terms of memory

## Server Cache Cons

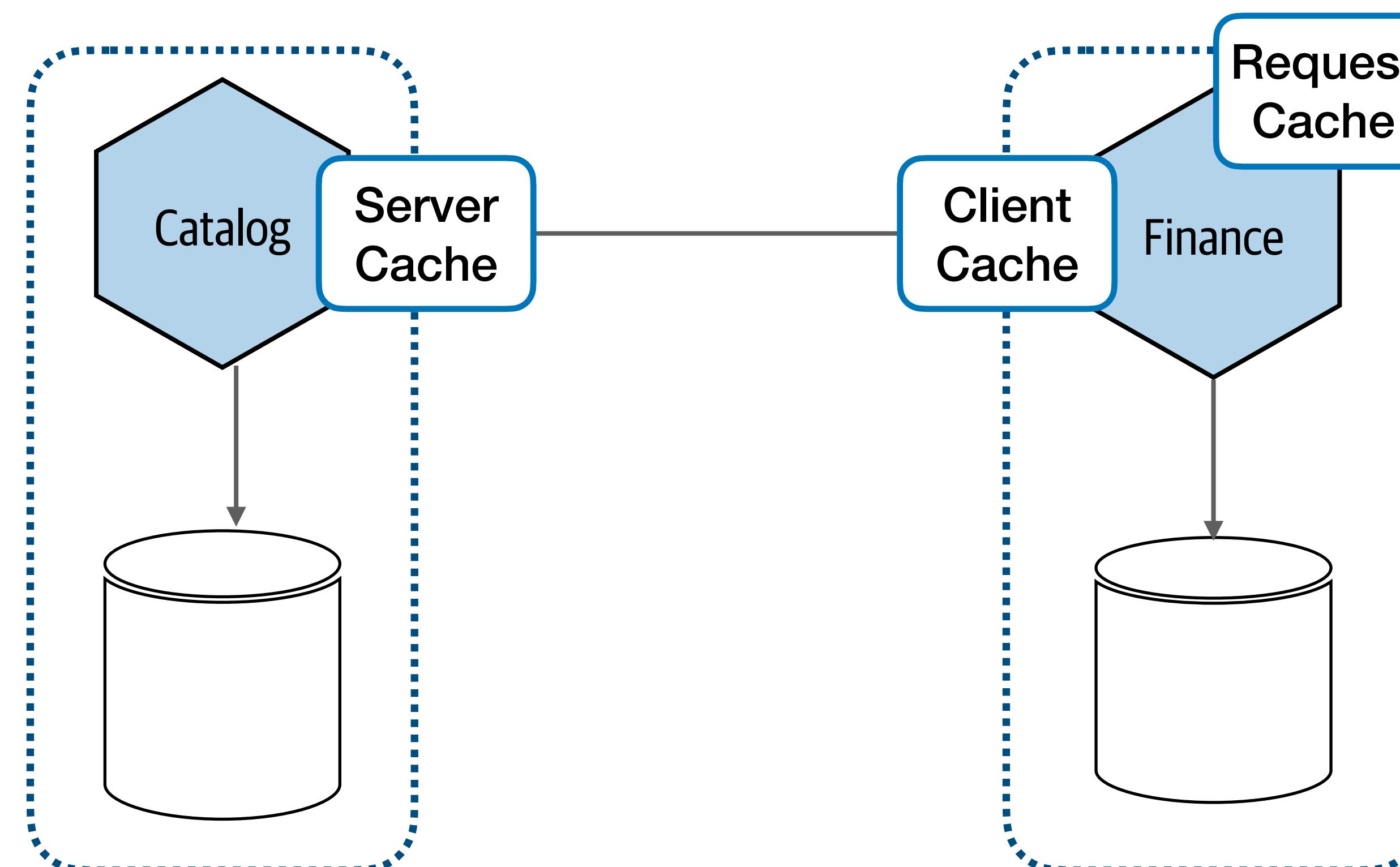
Call still needs to be made



# WHERE COULD WE CACHE?

## Request Cache Pros

What are this week's best sellers?



## Server Cache Pros

- Invalidation easier
- Can be more efficient in terms of memory

## Server Cache Cons

- Call still needs to be made

## Client Cache Pros

- Avoids network call
- Could operate if Catalog was unreachable

## Client Cache Cons

- More clients = more memory, inconsistency
- Invalidation

# WHERE COULD WE CACHE?

## Request Cache Pros

Super efficient

What are this week's  
best sellers?

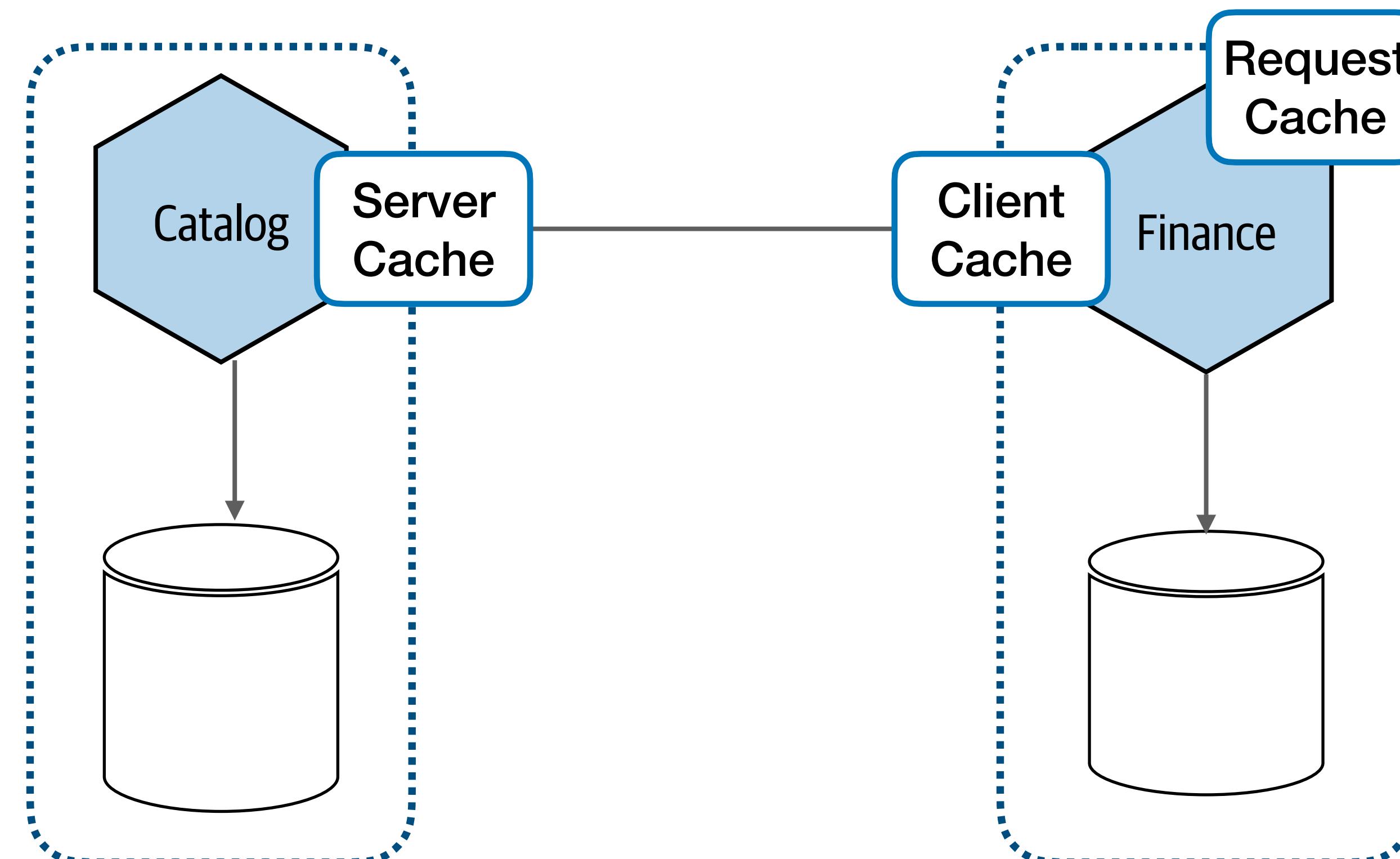
## Server Cache Pros

Invalidation easier

Can be more efficient in  
terms of memory

## Server Cache Cons

Call still needs to be made



## Client Cache Pros

Avoids network call

Could operate if Catalog was  
unreachable

## Client Cache Cons

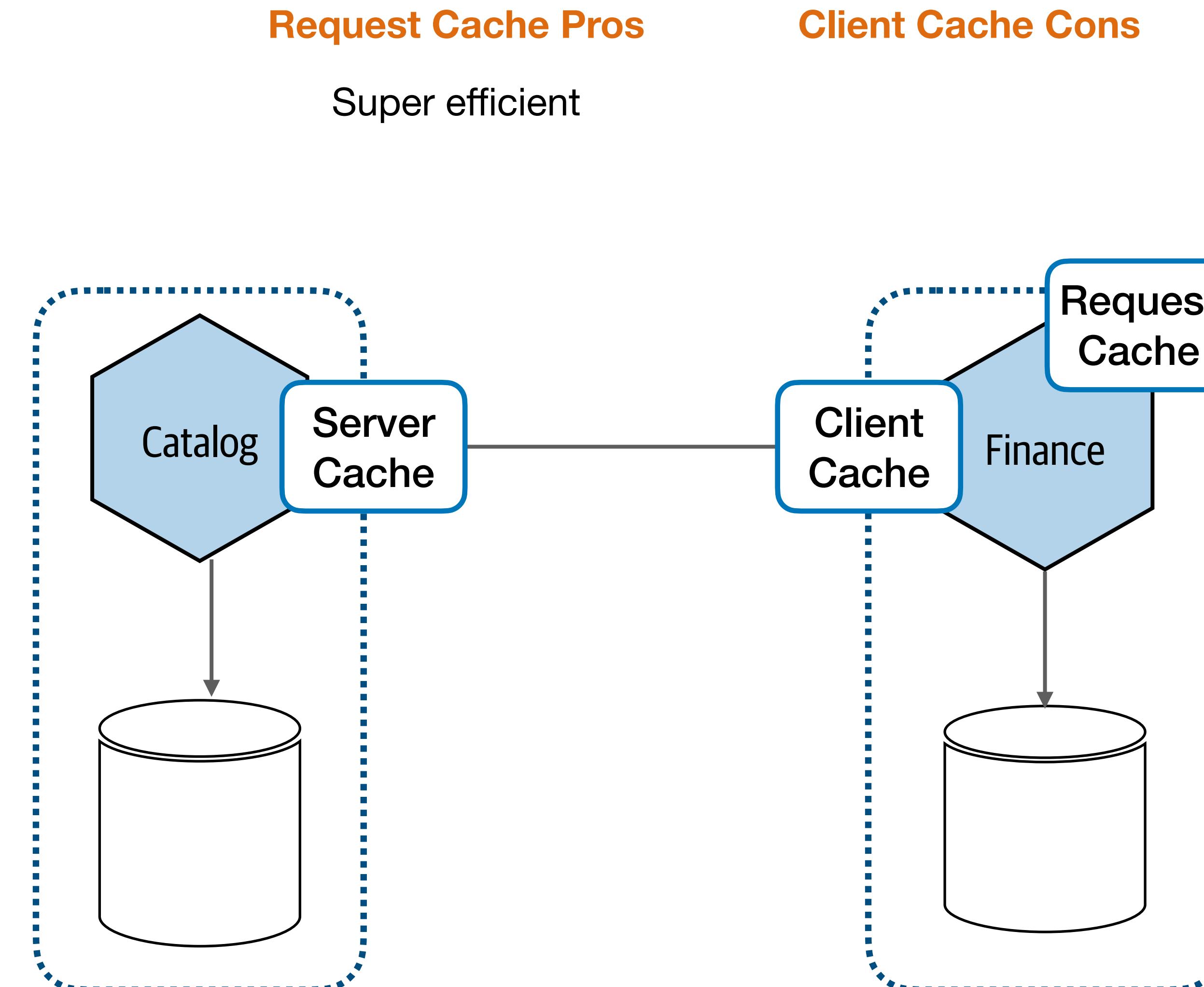
More clients =  
more memory, inconsistency

Invalidation

# WHERE COULD WE CACHE?

**Server Cache Pros**  
Invalidation easier  
Can be more efficient in terms of memory

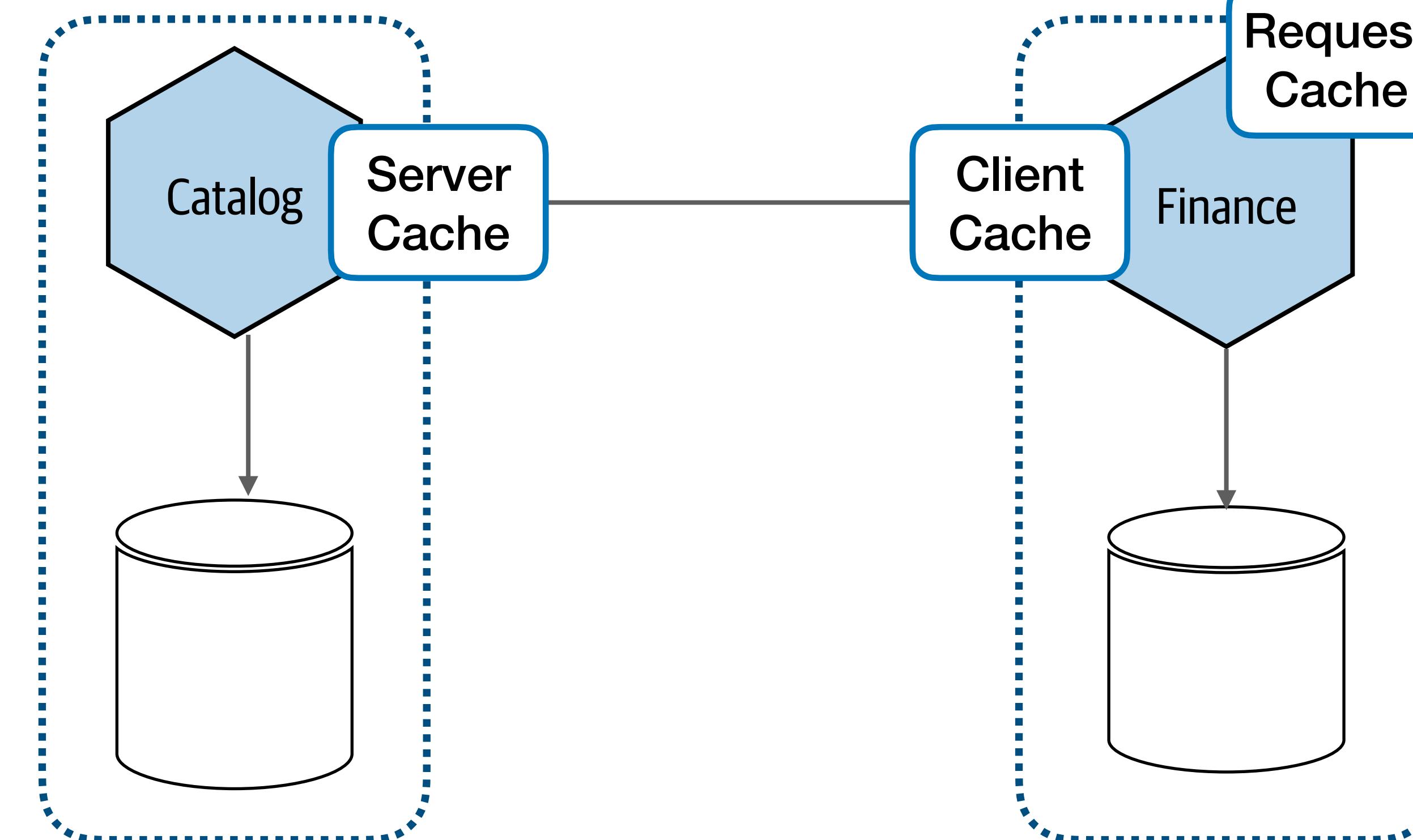
**Server Cache Cons**  
Call still needs to be made



# WHERE COULD WE CACHE?

**Server Cache Pros**  
Invalidation easier  
Can be more efficient in terms of memory

**Server Cache Cons**  
Call still needs to be made



## Request Cache Pros

Super efficient

## Client Cache Cons

Highly-specific cache

**What are this week's best sellers?**

## Client Cache Pros

Avoids network call

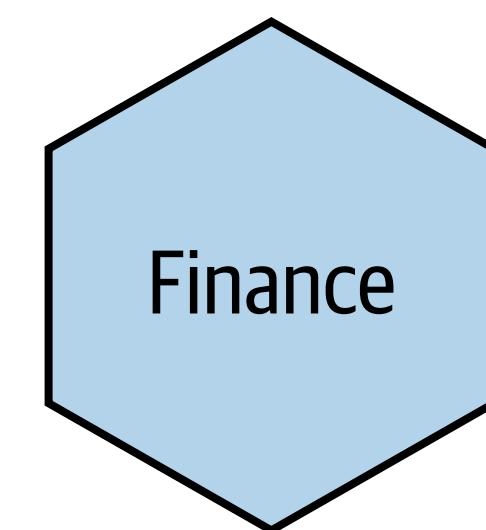
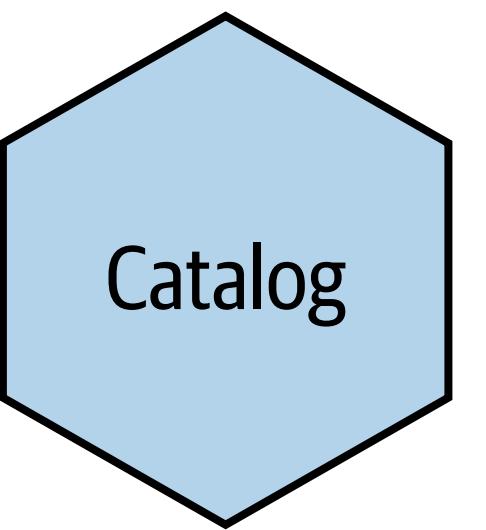
Could operate if Catalog was unreachable

## Client Cache Cons

More clients = more memory, inconsistency

Invalidation

# A DEDICATED SERVICE?



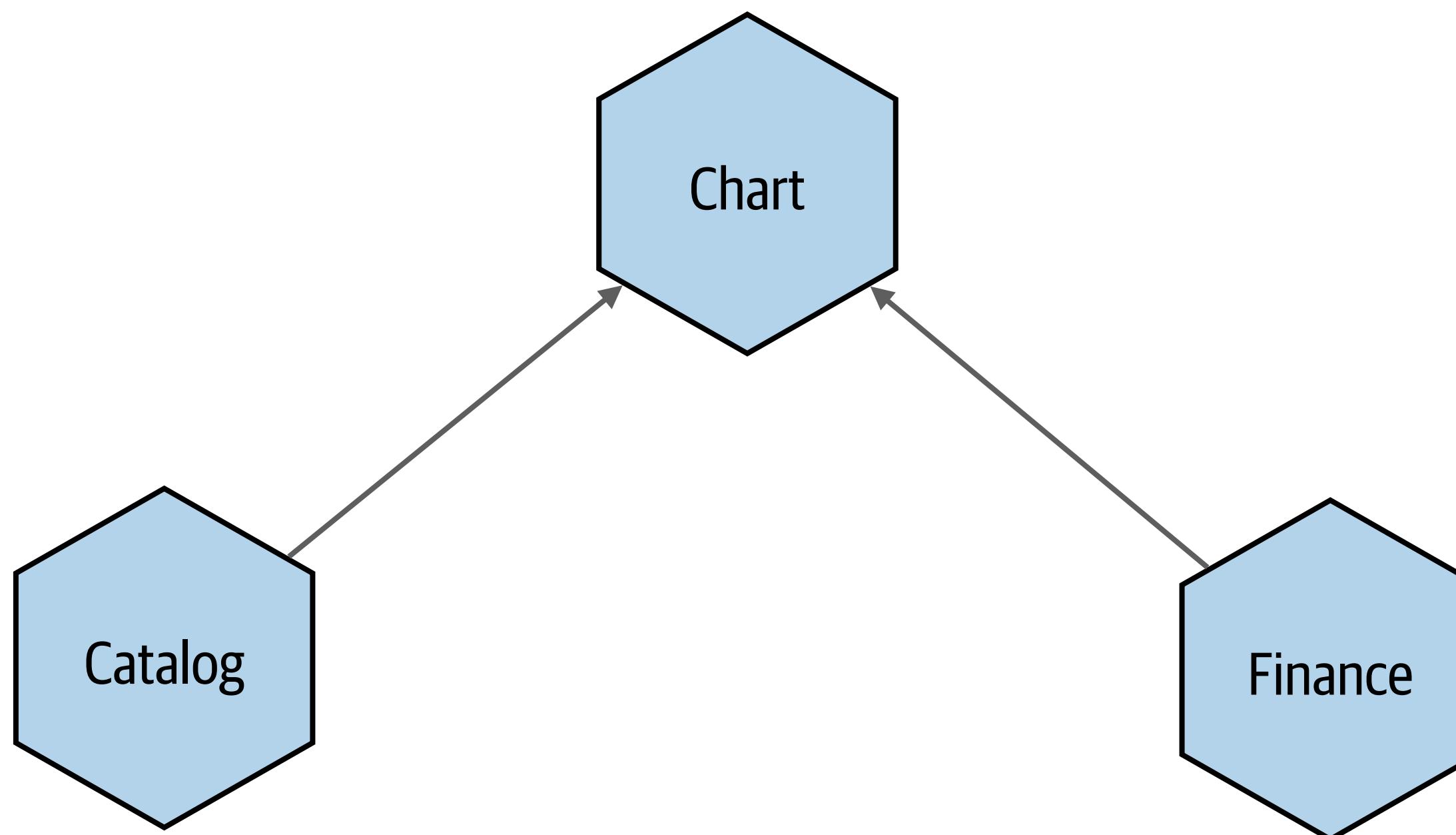
# A DEDICATED SERVICE?

Catalog

Chart

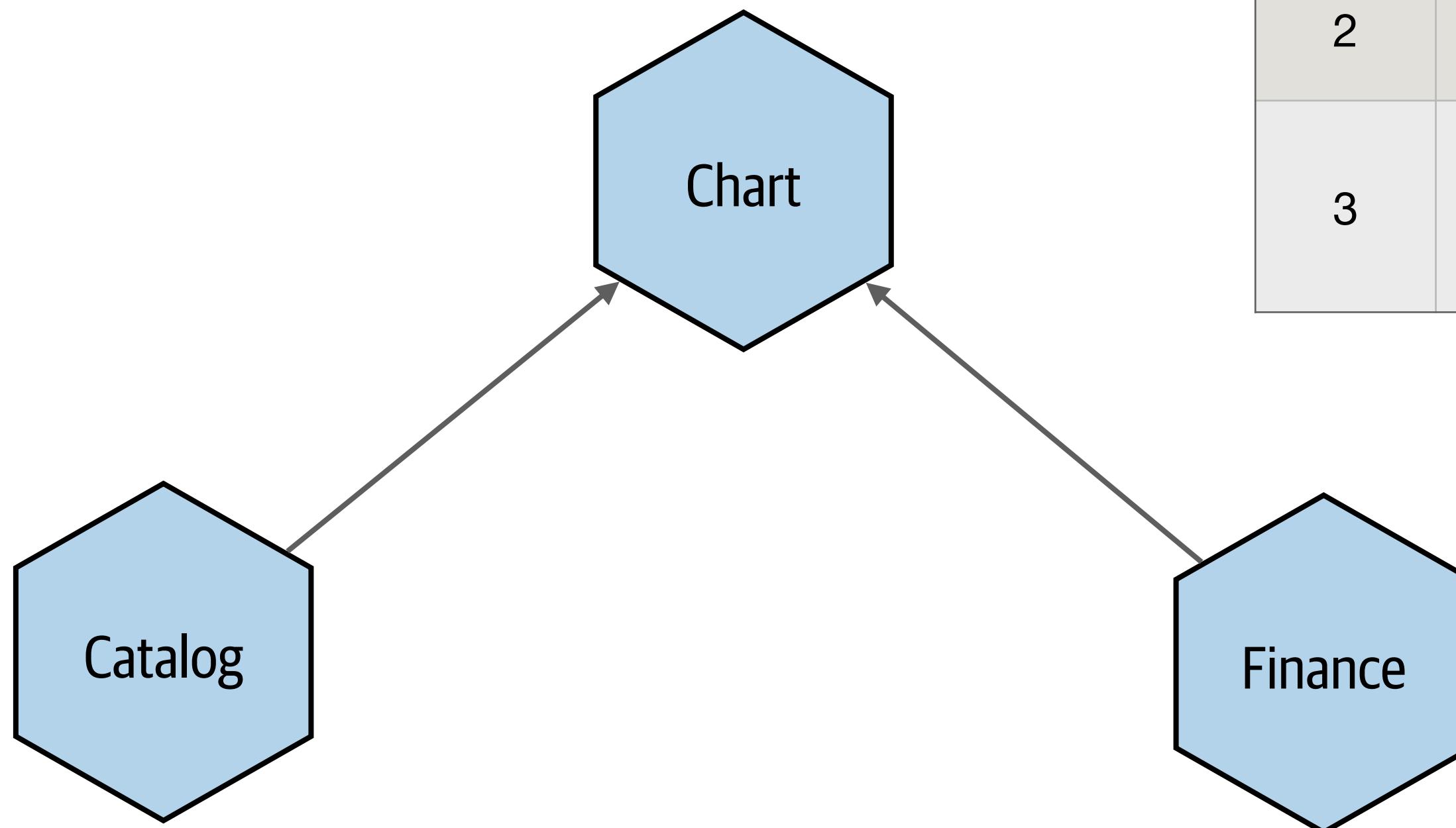
Finance

## A DEDICATED SERVICE?



# A DEDICATED SERVICE?

**Best Sellers This Week!**

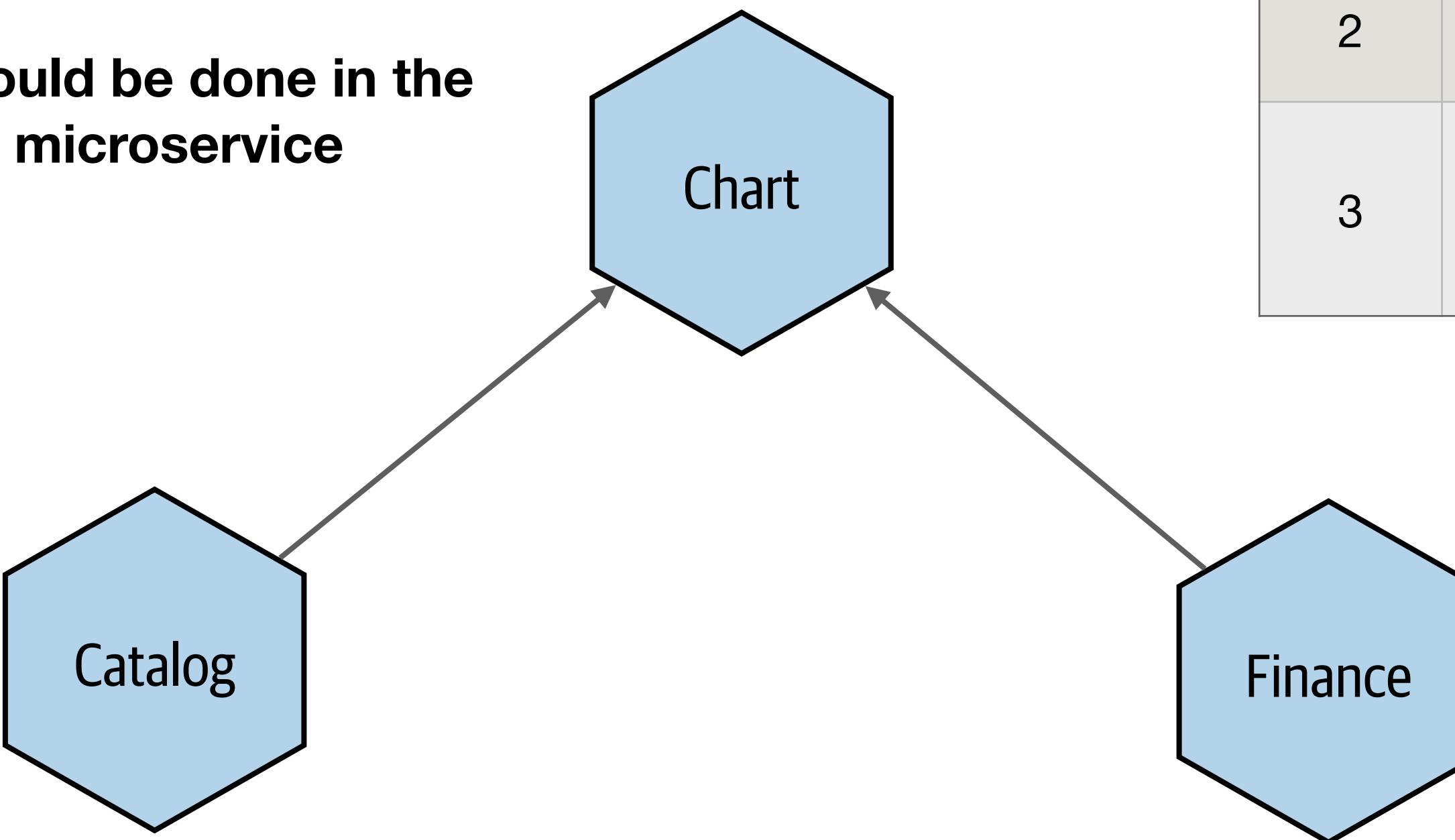


1	Death Polka, Vol 4
2	Greatest Hits of Peter Andre
3	Now That's What I Call Hoovercore

# A DEDICATED SERVICE?

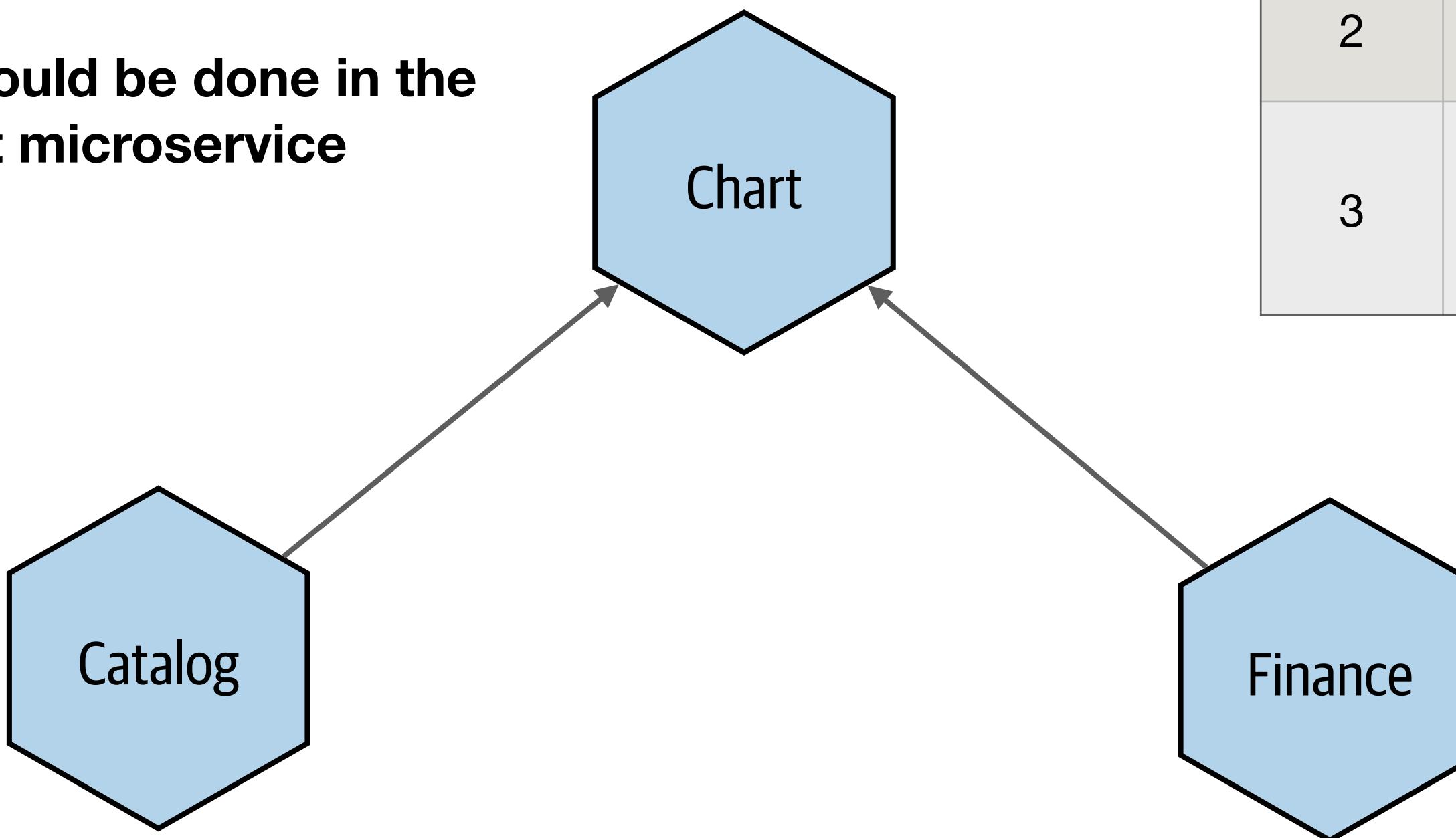
**Best Sellers This Week!**

**Caching could be done in the  
Chart microservice**



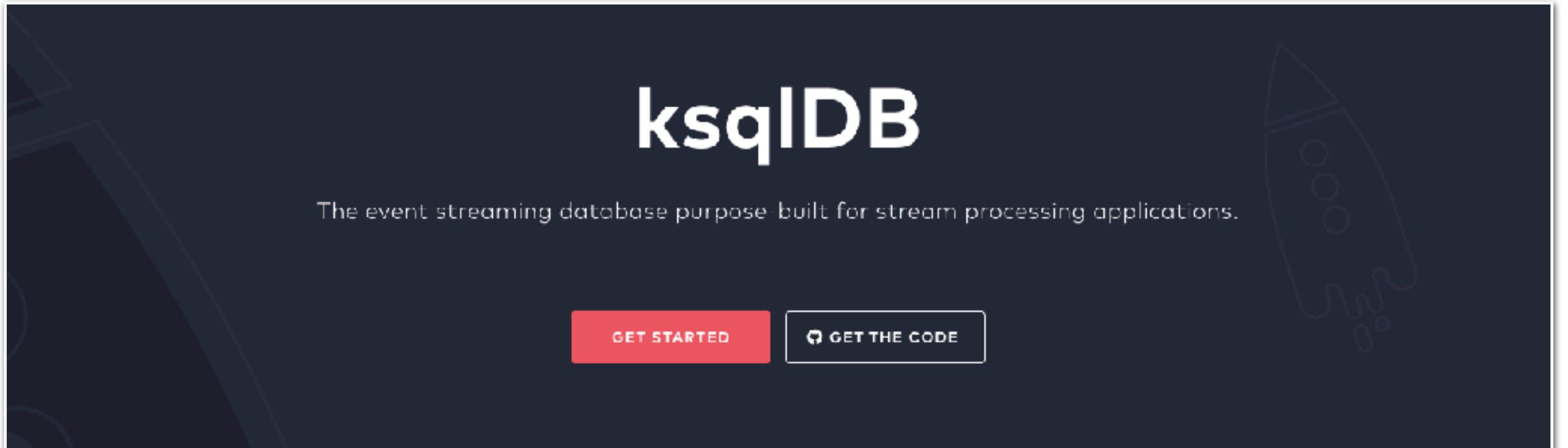
# A DEDICATED SERVICE?

**Caching could be done in the Chart microservice**



**Could also do a “live” query to the downstream microservices**

# KSQLDB



The event streaming database purpose built for stream processing applications.

[GET STARTED](#) [GET THE CODE](#)

**Real, real-time**  
Build applications that respond immediately to events. Craft materialized views over streams. Receive real-time push updates, or pull current state on demand.

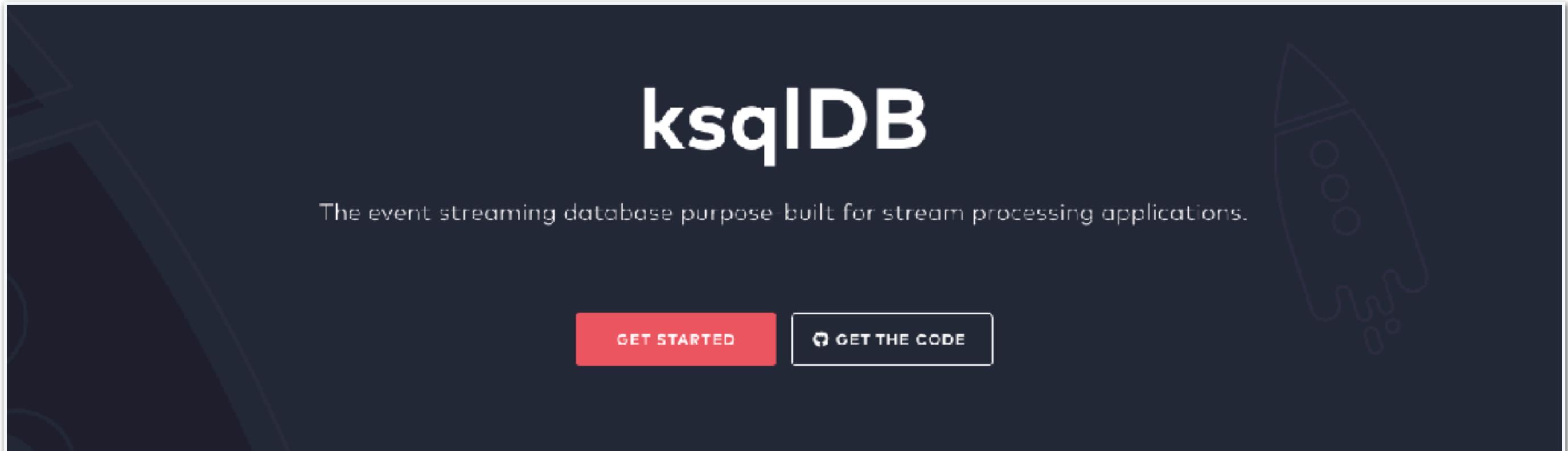
**Kafka-native**  
Seamlessly leverage your existing [Apache Kafka®](#) infrastructure to deploy stream-processing workloads and bring powerful new capabilities to your applications.

**What, not how**  
Use a familiar, lightweight syntax to pack a powerful punch. Capture, process, and serve queries using only SQL. No other languages or services are required.

Thousands of organizations love & trust ksqlDB

<https://ksqldb.io/>



The screenshot shows the ksqlDB homepage. At the top, the word "ksqlDB" is written in a large, white, sans-serif font. Below it, a subtitle reads "The event streaming database purpose built for stream processing applications." Two buttons are visible: a red "GET STARTED" button and a white "GET THE CODE" button. The main content area is divided into three sections: "Real, real-time", "Kafka-native", and "What, not how". Each section contains a brief description and a small icon. At the bottom, a light gray footer bar features the text "Thousands of organizations love & trust ksqlDB" and logos for Mailchimp, Bosch, Derivco, and Pushowl.

**Real, real-time**

Build applications that respond immediately to events. Craft materialized views over streams. Receive real-time push updates, or pull current state on demand.

**Kafka-native**

Seamlessly leverage your existing [Apache Kafka®](#) infrastructure to deploy stream-processing workloads and bring powerful new capabilities to your applications.

**What, not how**

Use a familiar, lightweight syntax to pack a powerful punch. Capture, process, and serve queries using only SQL. No other languages or services are required.

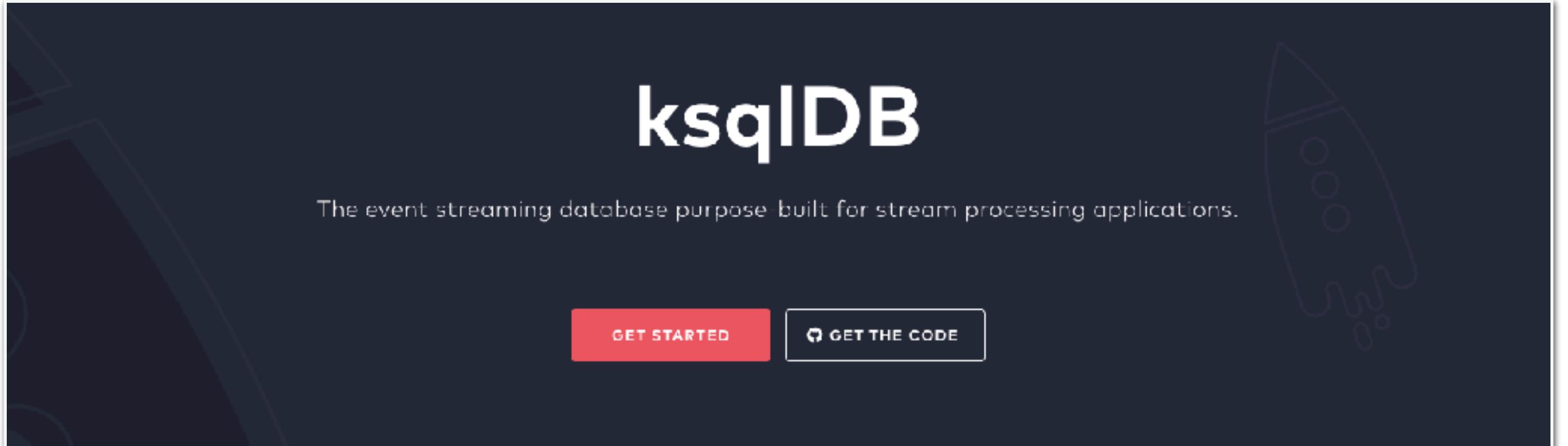
Thousands of organizations love & trust ksqlDB

<https://ksqldb.io/>

Apply SQL-style queries  
to streams being  
emitted from Kafka

# KSQLDB



The screenshot shows the ksqlDB homepage. At the top, the word "ksqlDB" is written in a large, white, sans-serif font. Below it, a subtitle reads "The event streaming database purpose built for stream processing applications." Two buttons are visible: a red "GET STARTED" button and a white "GET THE CODE" button. The main content area is divided into three sections: "Real, real-time", "Kafka-native", and "What, not how". Each section contains a brief description and a small icon. At the bottom, a light gray bar features the text "Thousands of organizations love & trust ksqlDB" and logos for Mailchimp, Bosch, Derivco, and Pushowl.

**Real, real-time**

Build applications that respond immediately to events. Craft materialized views over streams. Receive real-time push updates, or pull current state on demand.

**Kafka-native**

Seamlessly leverage your existing Apache Kafka® infrastructure to deploy stream-processing workloads and bring powerful new capabilities to your applications.

**What, not how**

Use a familiar, lightweight syntax to pack a powerful punch. Capture, process, and serve queries using only SQL. No other languages or services are required.

Thousands of organizations love & trust ksqlDB

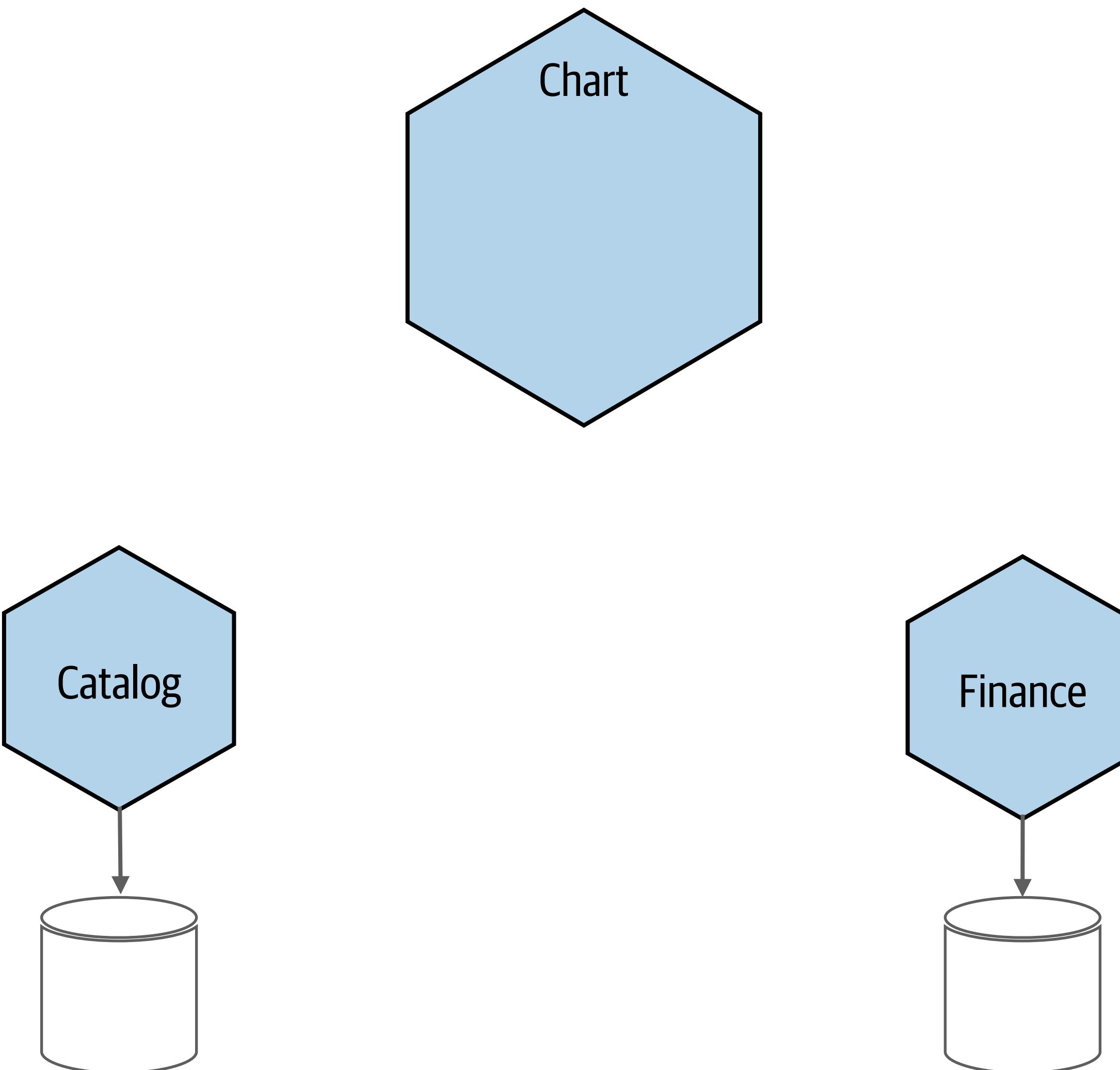
   

<https://ksqldb.io/>

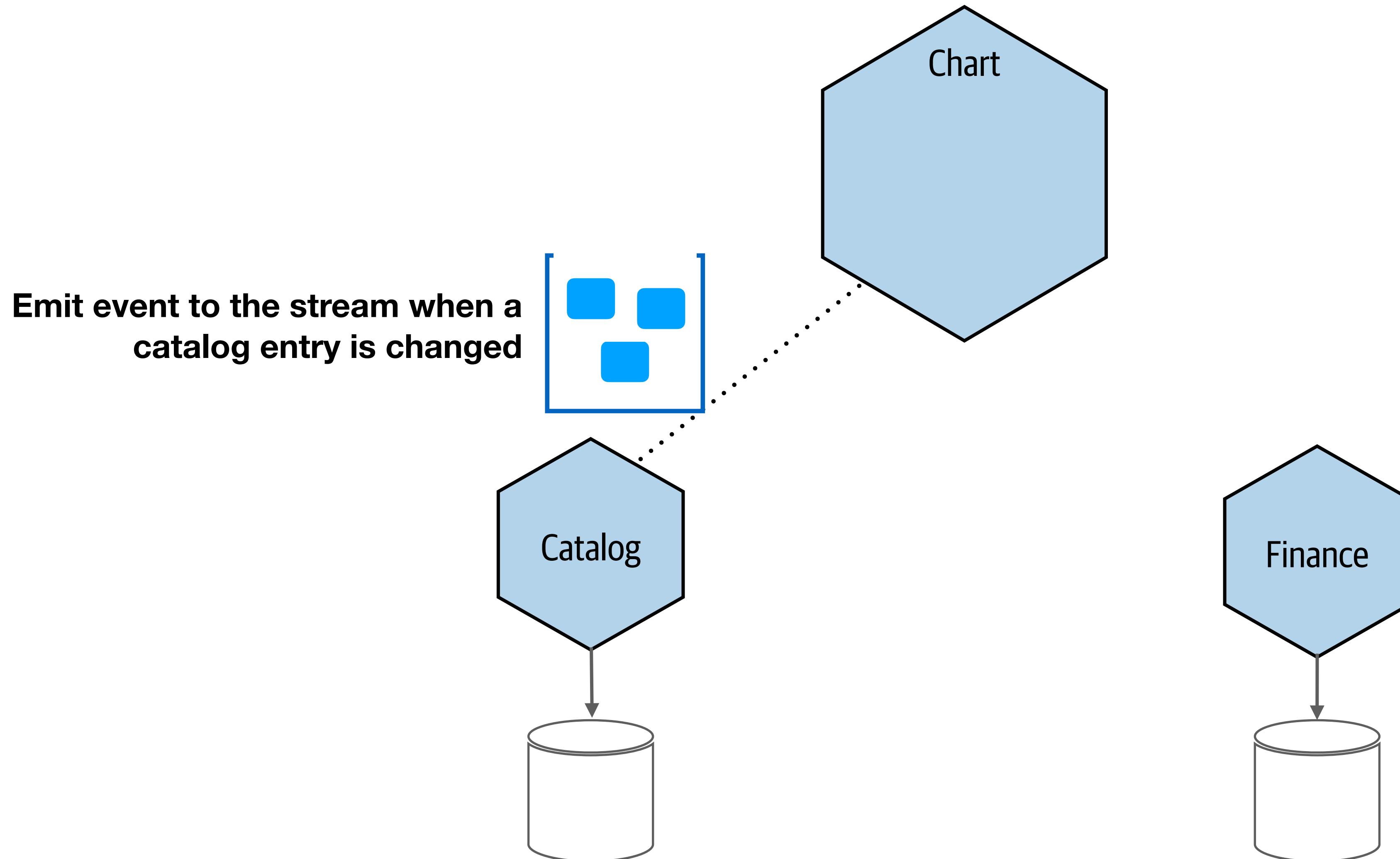
Apply SQL-style queries  
to streams being  
emitted from Kafka

Blends stream  
processing with  
standard SQL-use cases

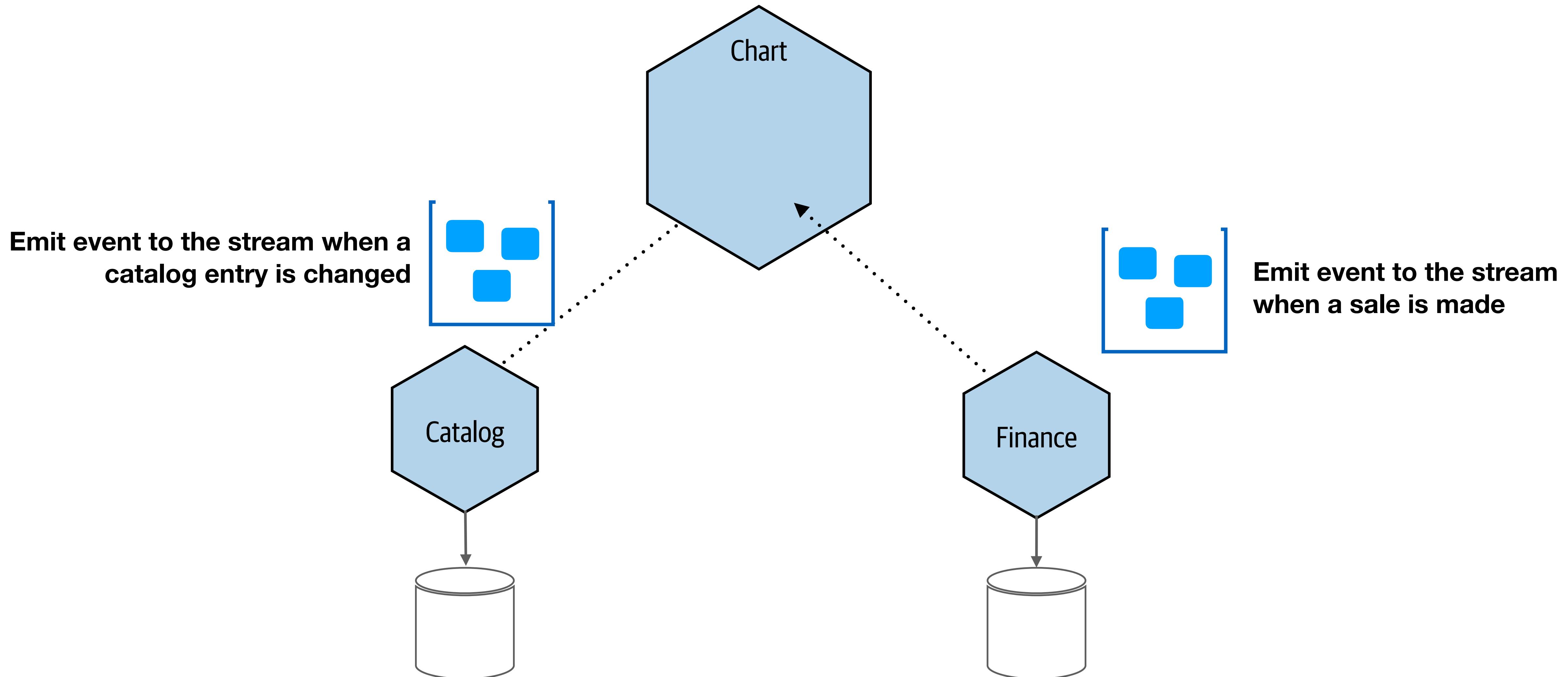
# KSQLDB EXAMPLE ARCHITECTURE



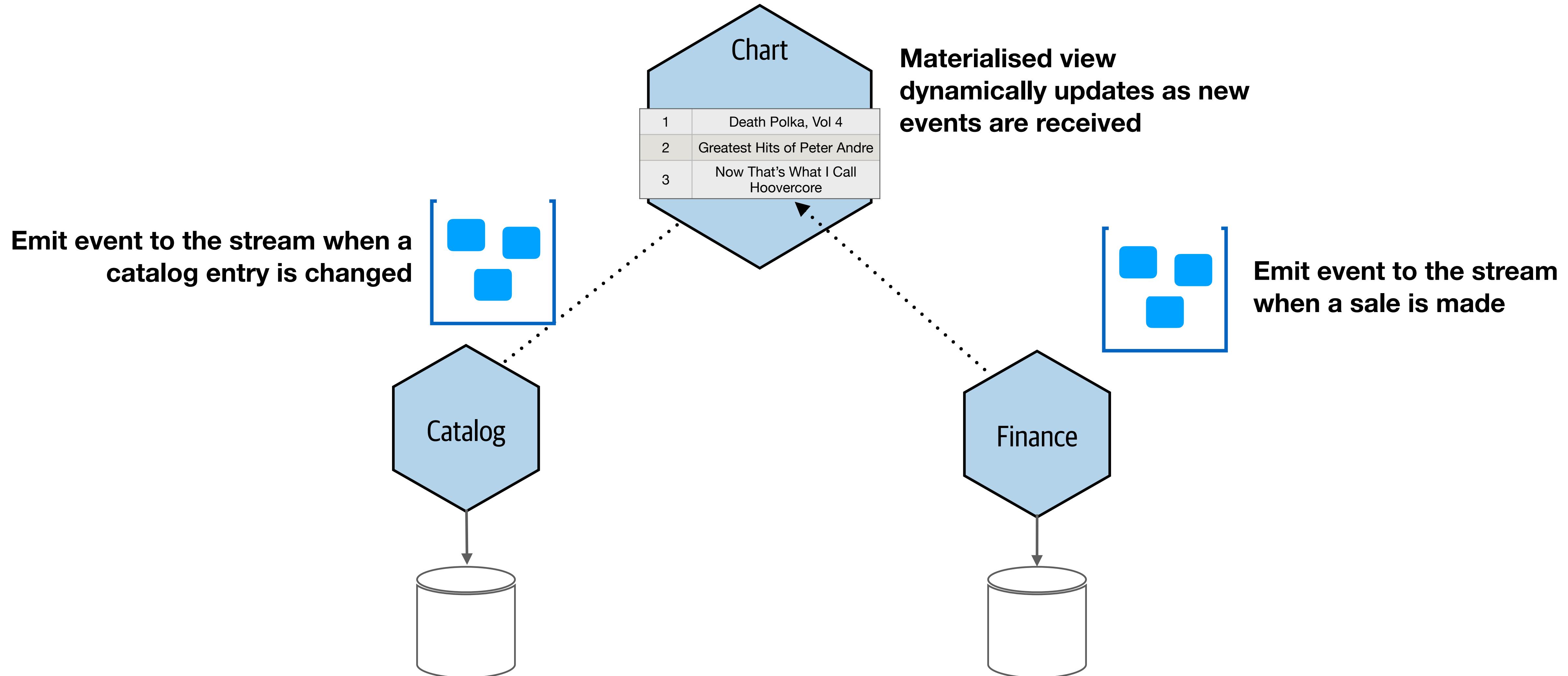
## KSQLDB EXAMPLE ARCHITECTURE



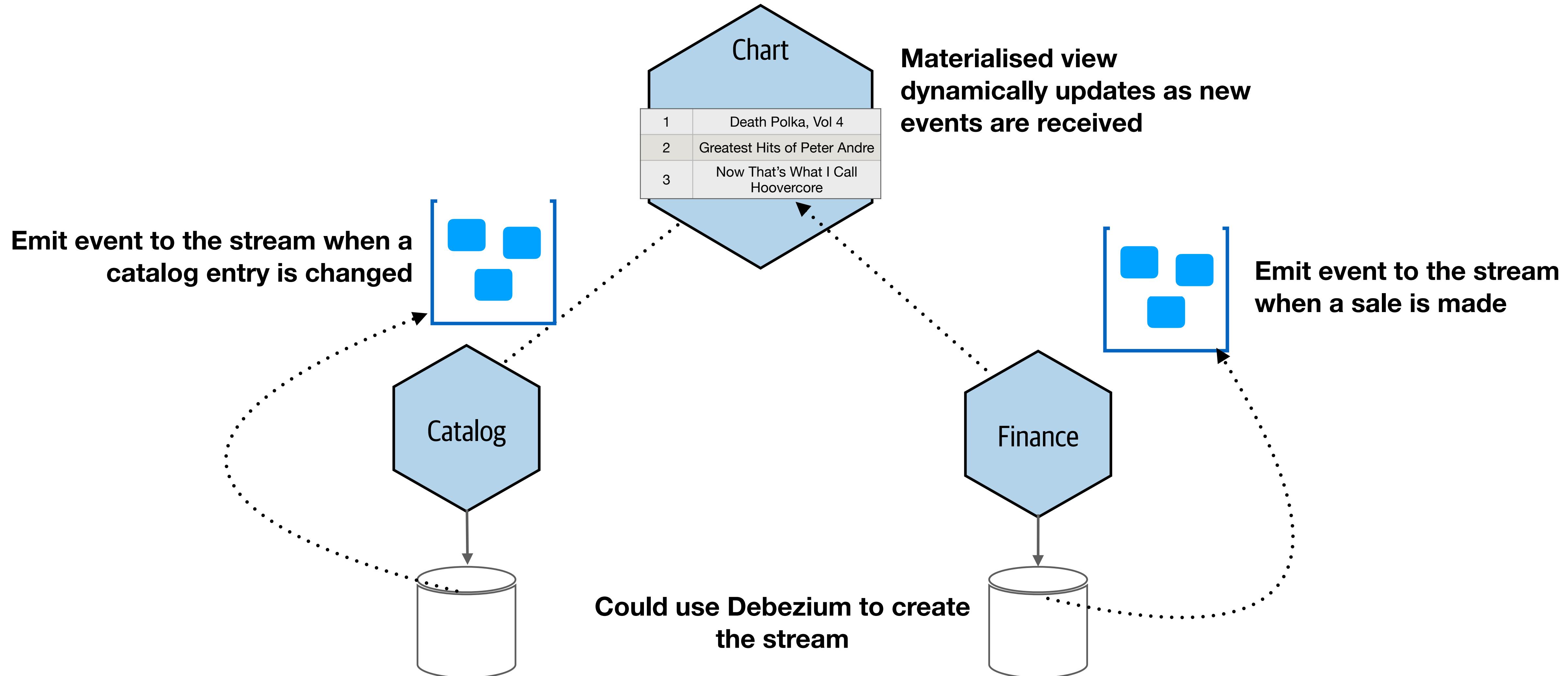
## KSQLDB EXAMPLE ARCHITECTURE



# KSQLDB EXAMPLE ARCHITECTURE



# KSQLDB EXAMPLE ARCHITECTURE



# THE CACHING PRIME DIRECTIVE

## THE CACHING PRIME DIRECTIVE

**Cache in as few places as possible**

## THE CACHING PRIME DIRECTIVE

**Cache in as few places as possible**

**Zero is the best number of caches**

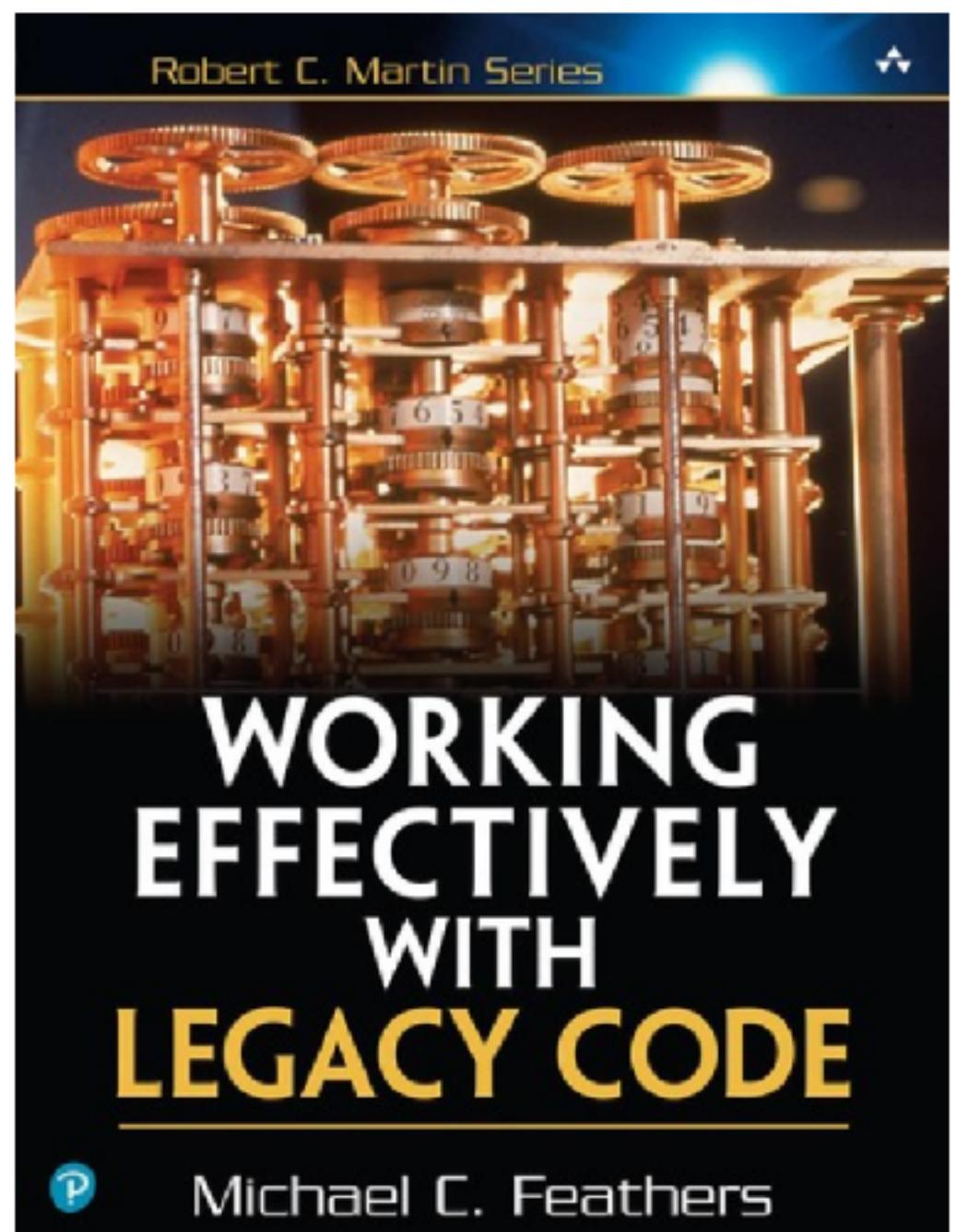
**POLL: AFTER ALL YOU'VE LEARNED, ARE YOU MORE OR LESS LIKELY TO USE MICROSERVICES THAN YOU WERE BEFORE?**

I'm much more likely to use them!

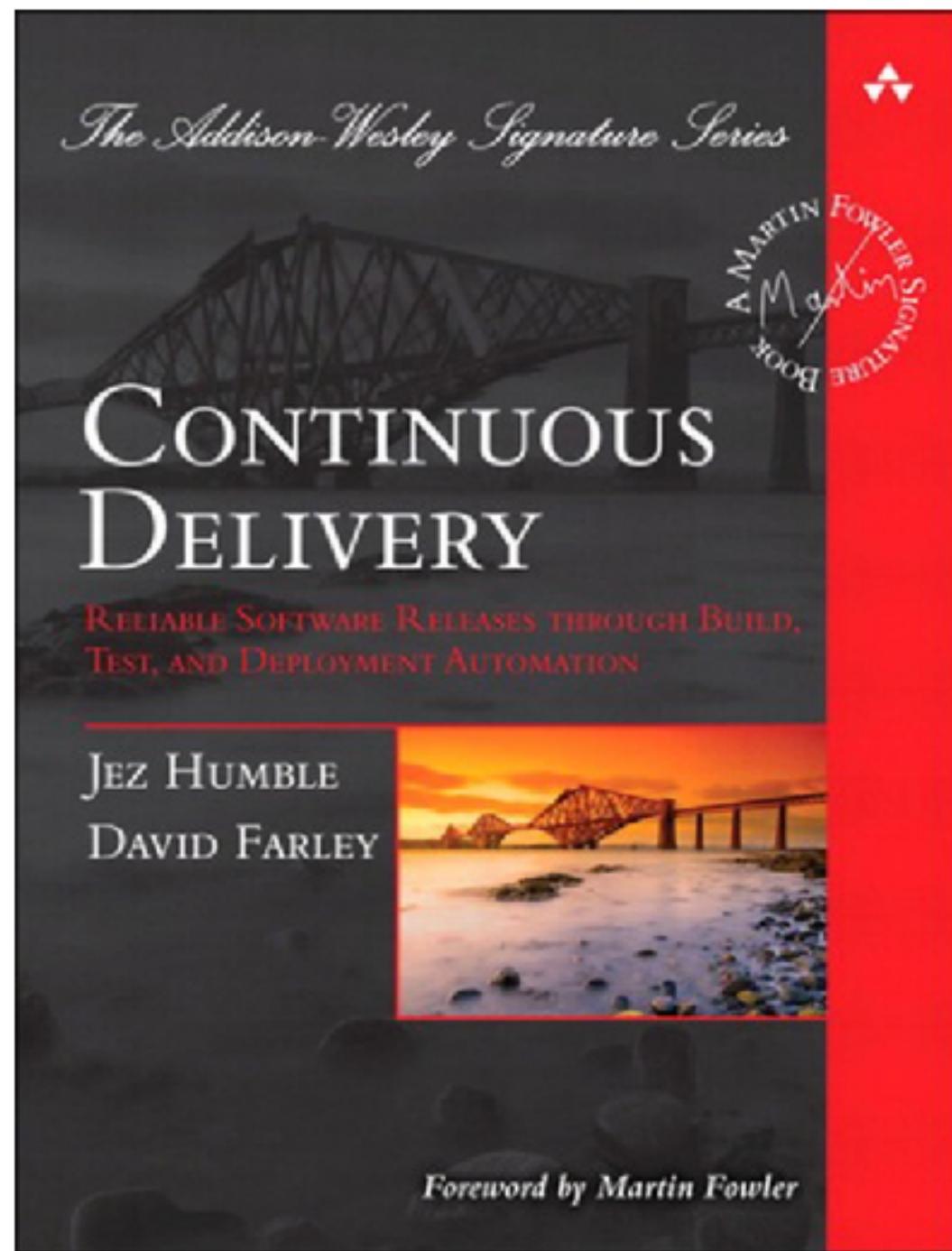
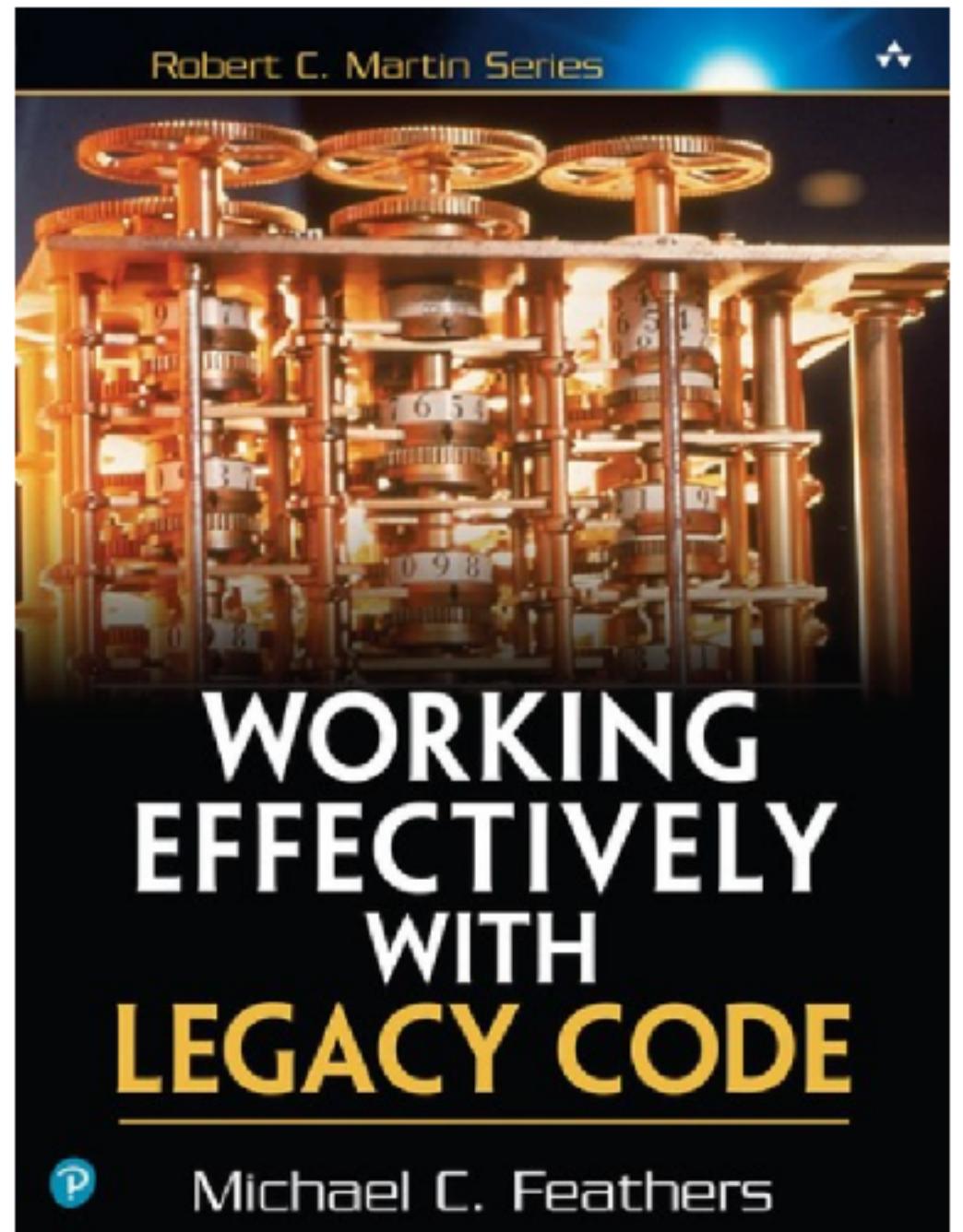
About the same

Less likely

## FURTHER READING

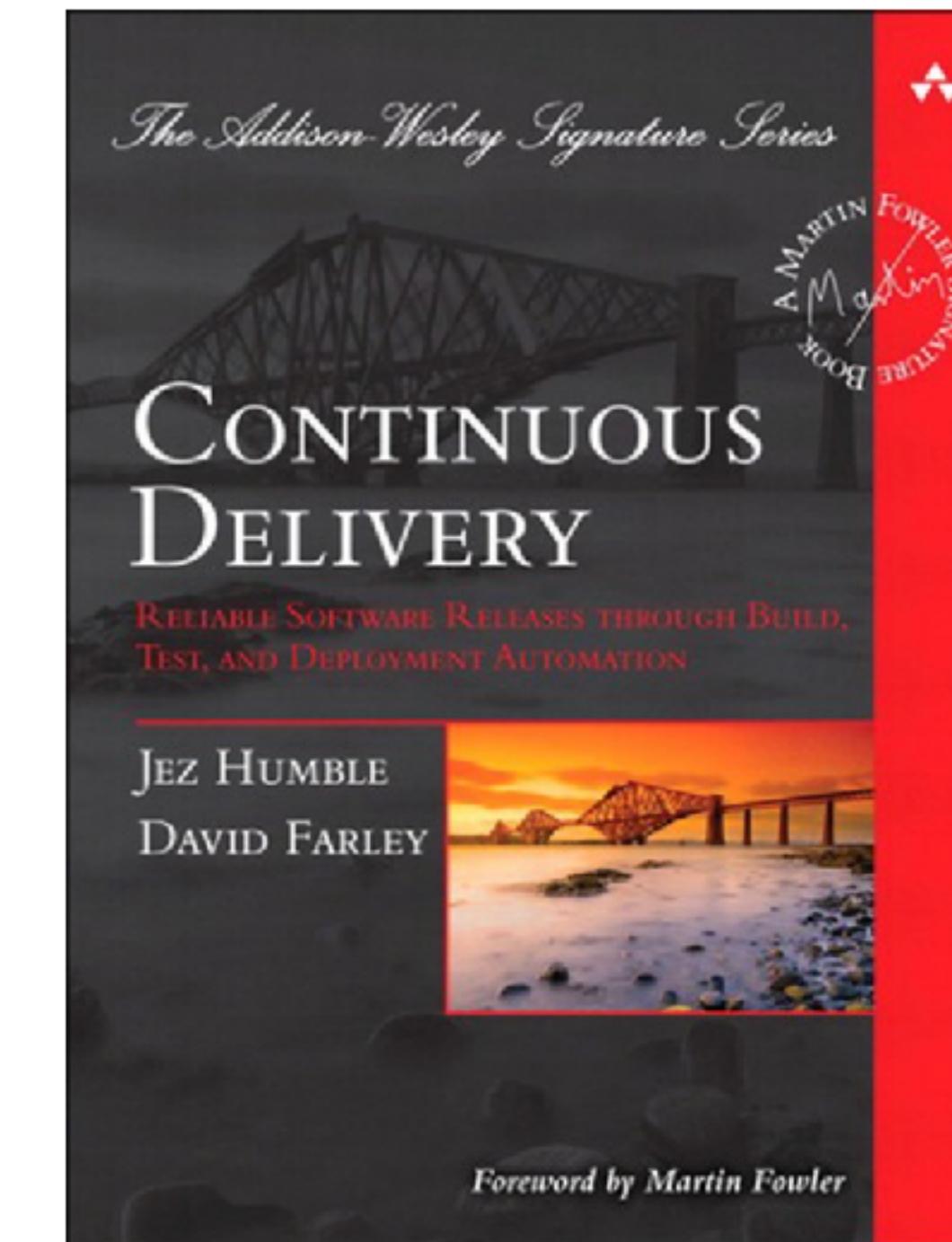
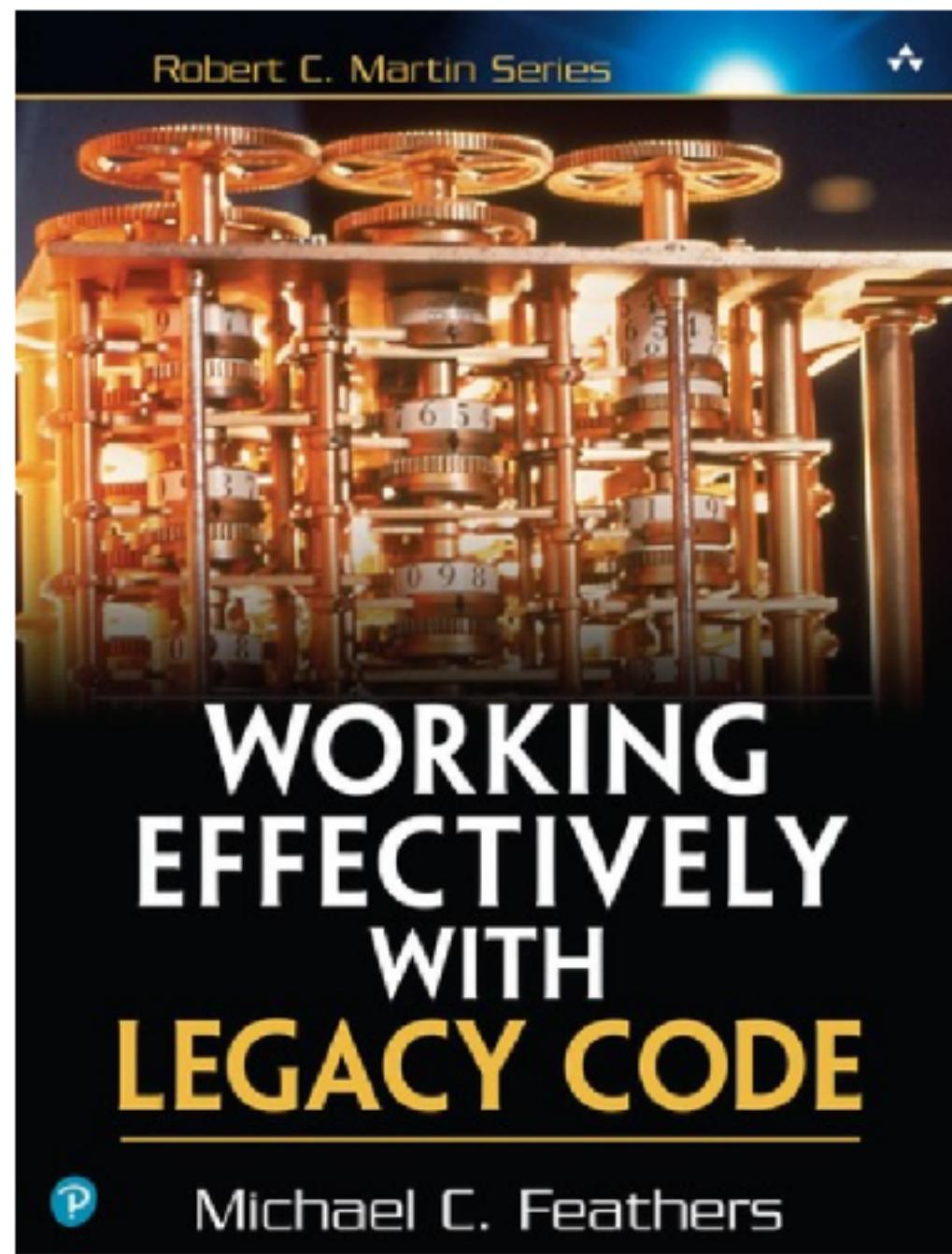


## FURTHER READING

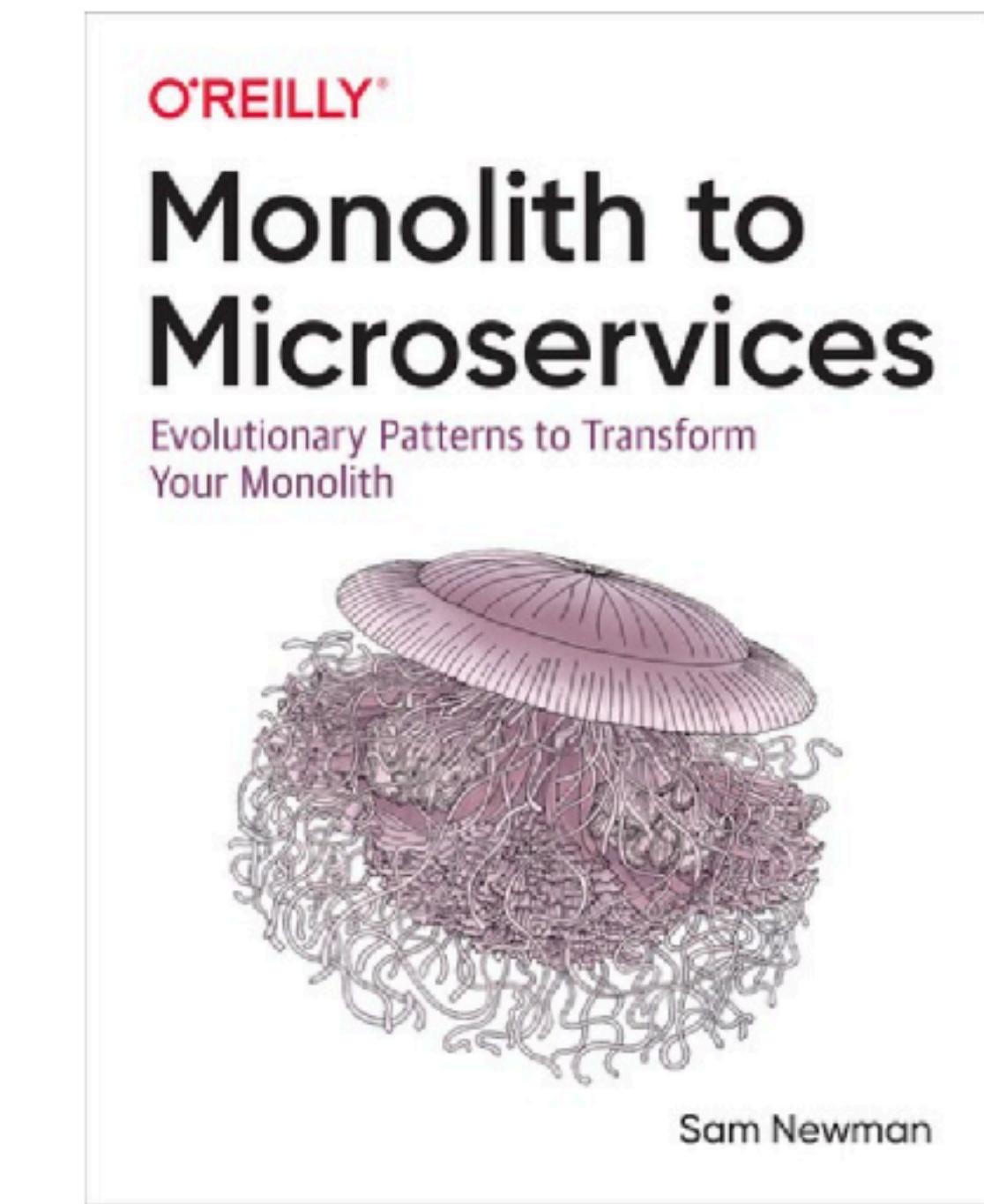


### Chapter 12: Managing Data

## FURTHER READING

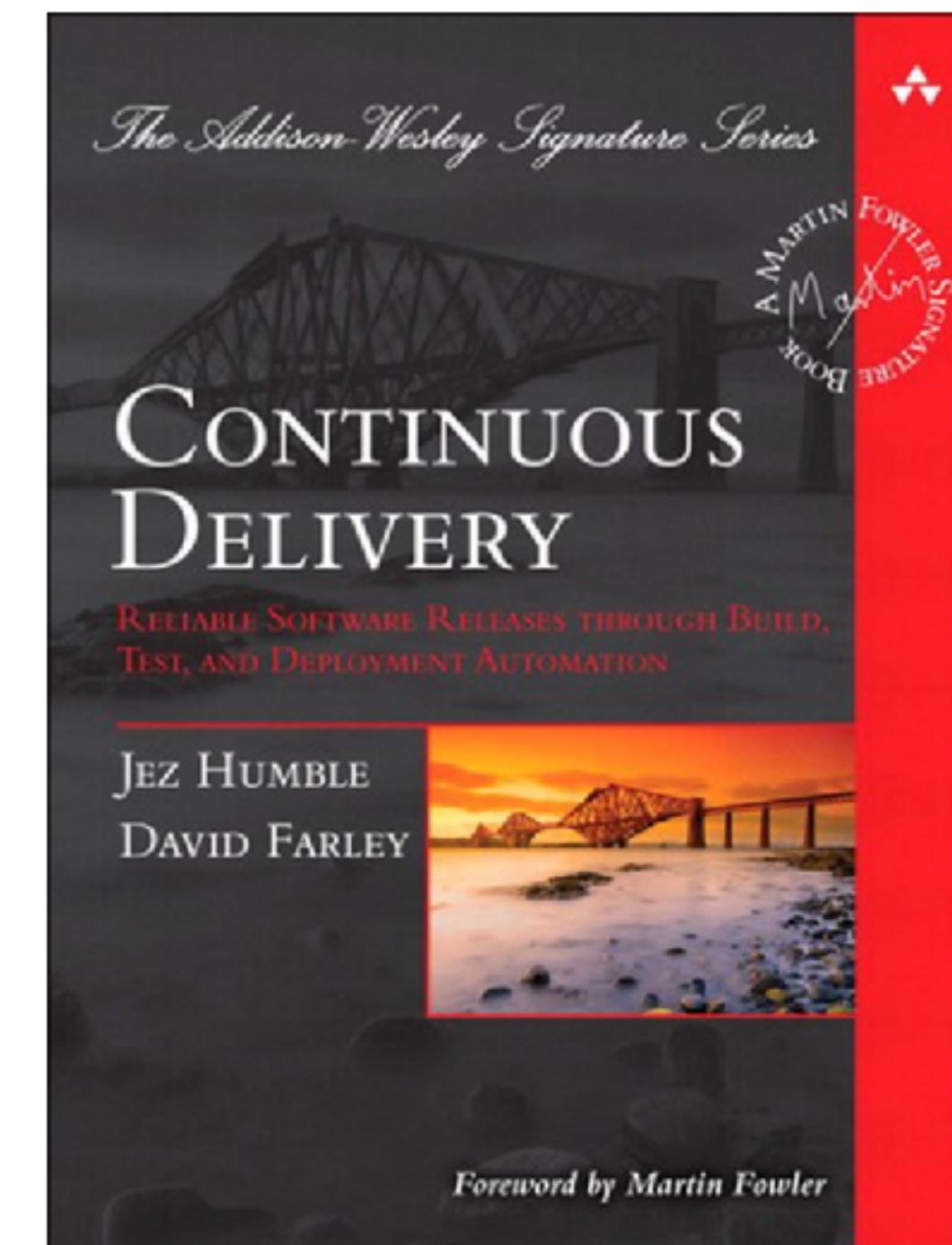
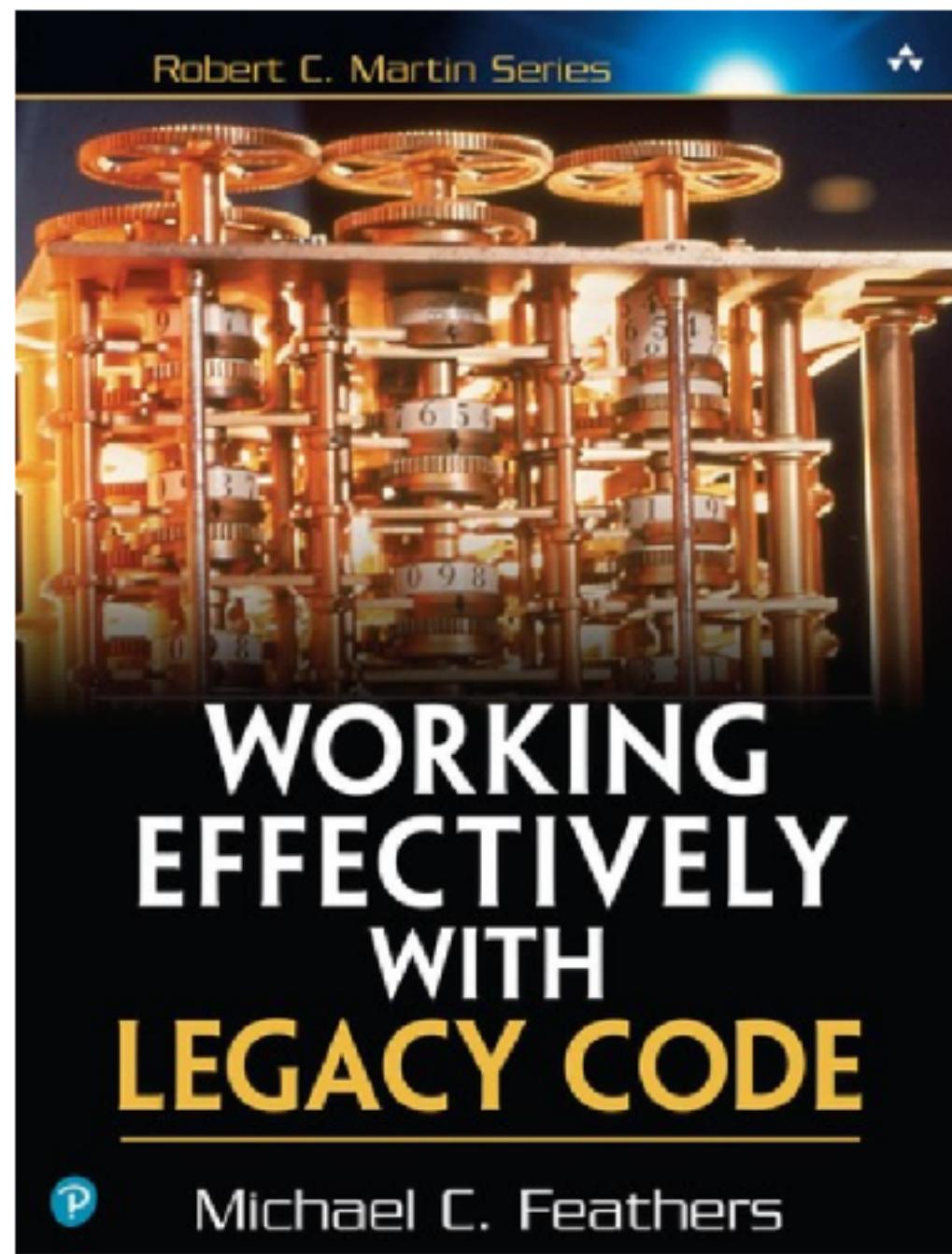


### Chapter 12: Managing Data

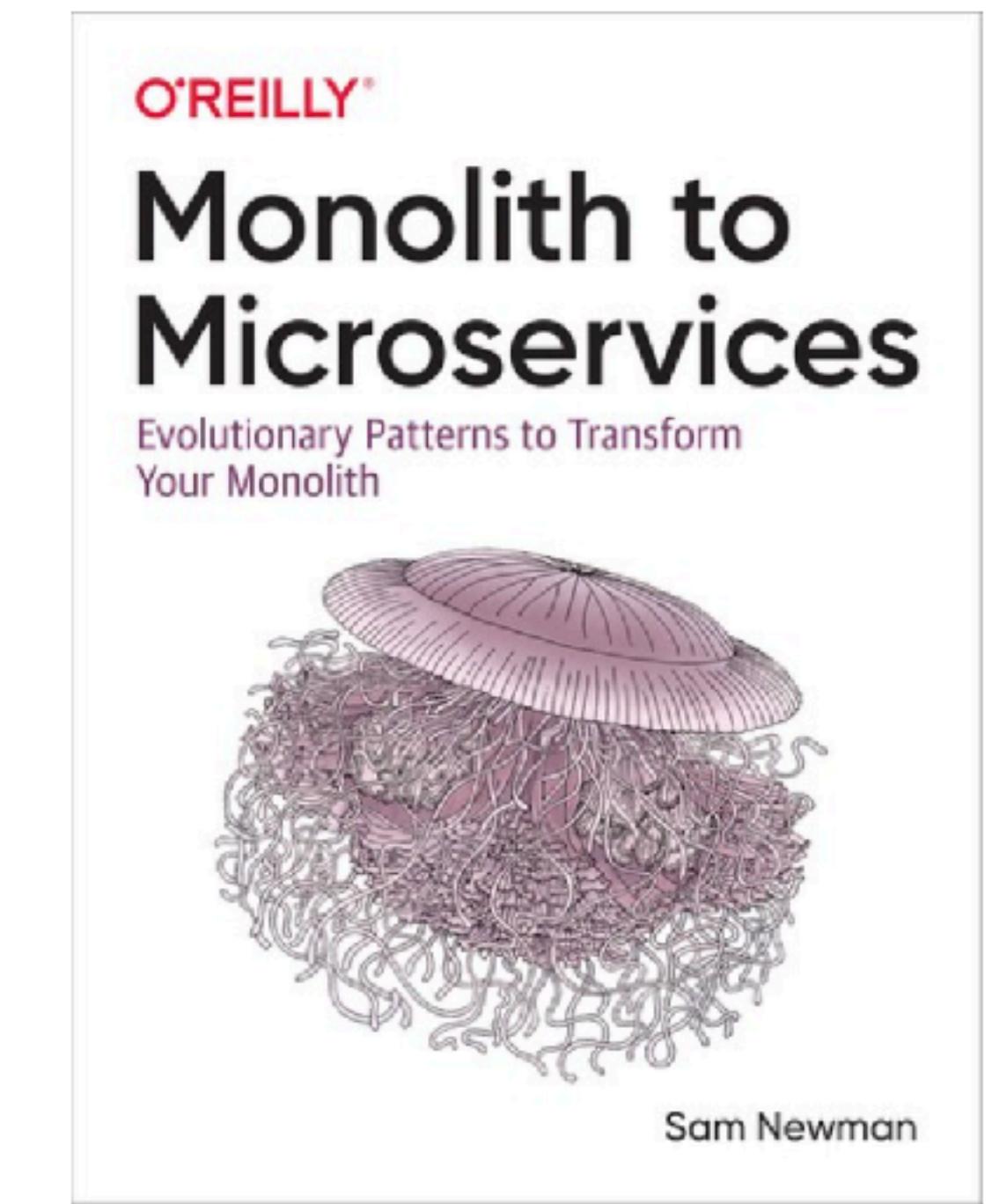


### Chapter 4: Decomposing the Database

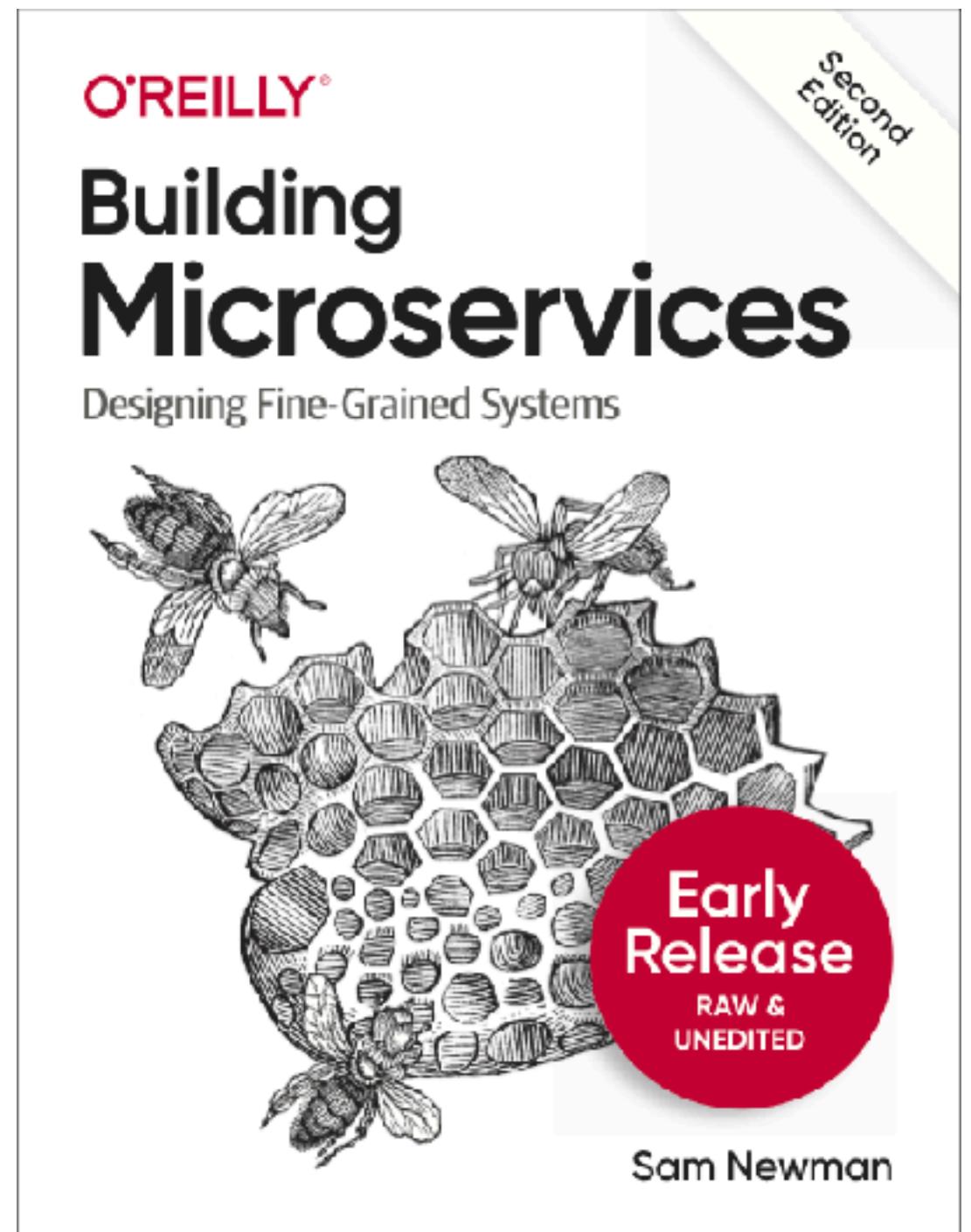
## FURTHER READING



Chapter 12: Managing Data

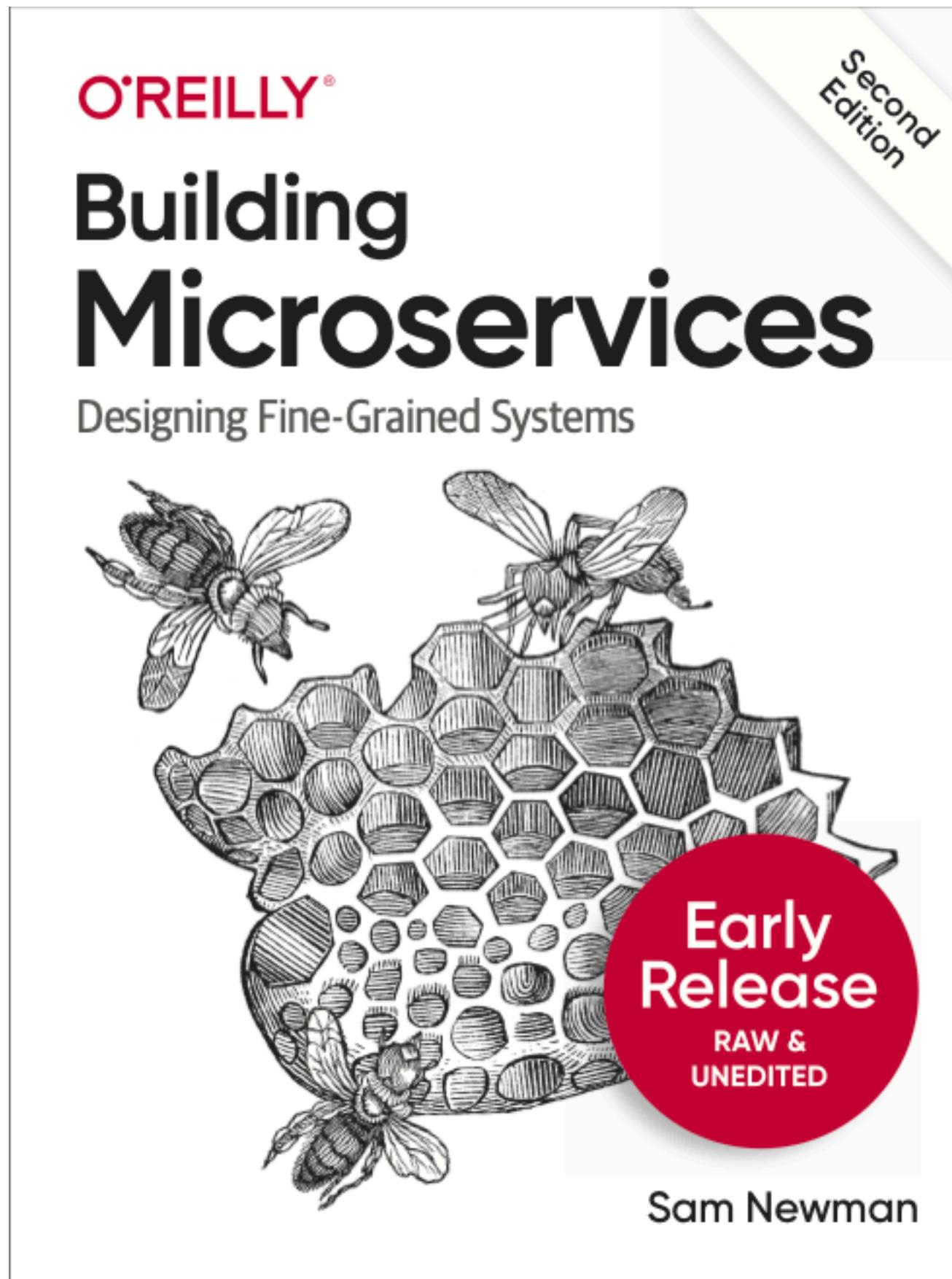


Chapter 4: Decomposing the  
Database



Chapter 8: Deployment

THANKS!



The screenshot shows the homepage of the Sam Newman website. At the top, the company logo is displayed. Below it, a navigation bar includes links for Home, About, Talks (which is highlighted in yellow), Podcast, Writing, and Contact. The main content area features a section titled 'Talks & Workshops.' with a sub-section titled 'What Is This Cloud Native Thing Anyway?'. This section includes a small diagram illustrating the relationship between Agile/Lean, Continuous Delivery, DevOps, Cloud Native, and Microservices. To the right of the main content, there is a sidebar with a 'Book!' section featuring the book cover of 'Building Microservices' and a 'Video!' section with a thumbnail image.

**Sam Newman.**

Home   About   **Talks**   Podcast   Writing   Contact

**Talks & Workshops.**

Here are a list of the talks I am currently presenting. On request, I can present different topics or even my older talks. If you want me to present these topics at your conference or company, then please contact me. You can also see where I'll be speaking next on my [events](#) page.

**What Is This Cloud Native Thing Anyway?**

45min Talk

A talk exploring what the hell Cloud Native means

→ [Find Out More](#)

**Feature Branches And Toggles In A Post-GitHub World.**

I have written a book called "Building Microservices", which is available now. Want to know more?  
→ [Read on...](#)

**Book!**

**Video!**

@samnewman

<https://samnewman.io/>

@samnewman